Hochschule Darmstadt

- Fachbereich Informatik -

Prototypische Entwicklung eines Prozessmanipulators zur Flexibilisierung von Geschäftsprozessinstanzen

Abschlussarbeit zur Erlangung des akademischen Grades
Master of Science (M.Sc.)

vorgelegt von

Matthias Küch

Referent:	*Prof. Dr. Urs Andelfinger*
Korreferent:	*Prof. Dr. Frank Bühler*
Betreuer:	*Dr. Harald Schöning*

Ausgabedatum:	*11.08.2010*
Abgabedatum:	*11.02.2011*

Erklärung

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

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, den 11. Februar 2011

Geheimhaltung

Diese Masterarbeit darf weder vollständig noch auszugsweise ohne schriftliche Zustimmung des Autors vervielfältigt, veröffentlicht oder Dritten zugänglich gemacht werden. Die abgegebenen Masterarbeiten werden von dem Hauptreferenten unter Verschluss gehalten. Mir ist bekannt, dass die Geheimhaltung nicht die Erstellung eines Masterposters sowie die Durchführung des Kolloquiums berührt. Die Geheimhaltungsverpflichtung erlischt automatisch nach 5 Jahren.

Darmstadt, den 11. Februar 2011

Abstrakt

IT-gestützte Geschäftsprozesse müssen heute flexibel auf Ereignisse der Außenwelt reagieren können. Das wird insbesondere nötig, da da in Prozessen nicht mehr alle möglichen Sonder- und Ausnahmefälle modelliert werden können. Durch die Forschungsgemeinschaft *Allianz digitaler Warenfluss* (ADiWa) werden Ansätze erforscht, um laufende Prozesse anhand von auftretenden Ereignissen zu flexibilisieren und an veränderte Gegebenheiten anzupassen. Als Teil von ADiWa ermittelt diese Arbeit, wie laufende Prozessinstanzen in der bestehenden Prozessausführungsumgebung von *Software AG webMethods* jederzeit, vollständig und ohne Vorbereitung flexibilisiert werden können. Flexibilisierungen sind z. B. das Hinzufügen, Löschen oder Überspringen von Prozessteilen. Die möglichen Flexibilisierungs-Arten sollten als frei wählbare Funktionen bereitgestellt und ereignis-gesteuert aufgerufen werden können.

Dazu mussten die theoretischen Grundlagen und die praktischen Möglichkeiten der ereignis-gesteuerten Geschäftsprozessflexibilisierung untersucht werden. Zunächst wurden die Themen „*Complex Event Processing*“ (CEP) und „*Event-Driven Business Process Management*“ (ED-BPM) eingeführt und die webMethods-Suite mit ihren Komponenten vorgestellt. Damit der Umfang der zu implementierenden Prozessflexibilitäts-Funktionen geklärt werden konnte, wurden Flexibilitäts-Taxonomien zur Klassifizierung und Bewertung ermittelt. Anschließend wurden Flexibilitäts-Muster der relevanten Flexibilitäts-Klassen „Prozess-Steuerung“, „Abweichung im Sequenzfluss“, „Veränderung“, und „Externalisierung“ vorgestellt. Als Ergebnis wurden Muster ermittelt, die Prozessinstanzen auf funktionaler, informationeller, operationaler und auf Verhaltens-Ebene flexibilisieren können. Diese Muster wurden im Hinblick auf Machbarkeit und Relevanz bewertet und anknüpfende Ansätze in der webMethods-Suite gesucht.

Die gefundenen Ansätze wurden im Prozessmanipulator mit unterschiedlichem Erfolg implementiert. Tritt ein Prozessflexibilisierungs-Ereignis auf, kann eine Instanz an einer bestimmten Stelle wiederholt, ihre Daten verändert oder eingeschränkt ihre Modellierung zur Laufzeit angepasst werden. Durch eine Integration von Prototyp, Prozessausführungsumgebung und Ereignisquelle ist es möglich, Prozessinstanzen ereignis-gesteuert zu flexibilisieren.

Abstract

Nowadays, IT-enabled business processes have to have the capacity to react to external business-related events. This is important for processes which are not modelled for all possible exceptional cases during the design phase. The research community "*Allianz digitaler Warenfluss*" (ADiWa), which is the basis for this thesis, is searching for approaches in order to make ongoing process instances more flexible and be better able to adapt to changing circumstances. As a part of ADiWa, this thesis discusses how ongoing process instances of the existing process engine of *Software AG webMethods* can be flexibilized at any time or place without excessive preparation. Examples of this increased flexibility include the ad-hoc introduction, deletion or skipping of different process components during runtime. The prototype has to be implemented as a set of flexibility functions providing event-driven flexibility.

In order to so, the theoretical and practical possibilities of improved flexibility for business processes have to be discussed. Firstly, the terms "*Complex Event Processing*" (CEP) and "*Event-Driven Business Process Management*" (ED-BPM) and the components of the webMethods Suite were introduced. To determine the scope of the implementation of flexibility functions, flexibility taxonomies were investigated. These taxonomies were later used for classifying and evaluating possible flexibility patterns. The flexibility classes "Process Control", "Deviation from sequence", "Change" and "Externalisation" were seen as relevant. As a result of this work, patterns were found which are able to improve flexibility in process instances on functional, informational, operational and behavioural layers of a business process. These patterns were tested to assess their feasibility and relevance.

These approaches were implemented with varying success in the process manipulator. If an event should occur which leads to flexibility, an instance can be repeated from a particular point, altered by its process data or remodelled with limitations during run-time. Through the integration of the prototype, the business process engine and the source of events, it is possible to increase flexibility in process instances driven by external events.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Programm-Auszüge

Kapitel 1

Motivation

1.1 Allianz Digitaler Warenfluss

Diese Arbeit ist in Zusammenarbeit mit der Software AG entstanden, die Teil der Forschungsgemeinschaft *Allianz Digitaler Warenfluss* (ADiWa)[1] ist. Das ADiWa-Konsortium besteht aus verschiedenen Unternehmen aus Wirtschaft und Wissenschaft. An ADiWa sind neben der Software AG (SAG) unter anderem SAP und IDS Scheer[2] als Technologiepartner sowie ABB, DB Schenker und die Handelskette Globus als Anwenderunternehmen beteiligt. Wissenschaftliche Partner sind neben dem Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI), mehrere Fraunhofer-Institute, die TU Darmstadt und die TU Dresden. Das Ziel dieser Forschungsgemeinschaft ist es, prototypische Anwendungen für die Anwendungspartner zu erforschen, umzusetzen und zu erproben. [ADi-2009b] ADiWa wird vom Bundesministerium für Bildung und Forschung gefördert. [Bun-2008] Das Projekt ist offiziell im Jahr 2009 gestartet und wird voraussichtlich im 4. Quartal 2011 abgeschlossen.

ADiWa untersucht mögliche Entwicklungen in Verbindung mit dem im „Internet der Dinge“[3]. Das Internet der Dinge besteht aus intelligenten physischen Objekten, die untereinander und mit Rechnersystemen kommunizieren können. Neue Technologien wie RFID-Chips[4] ermöglichen diese Kommunikation. Reale Objekte im Internet der Dinge sind z. B. Waren, die mithilfe von RFID-Chips ihren Standort und ihren Bestand mitteilen können. Zusätzlich werden Daten miteinbezogen, die der realen Welt entstammen. Solche Daten können beispielsweise Börsenkurse, Marktpreise oder Nachrichten sein. Beide Quellen von Daten, das Internet der Dinge und geschäftsrelevante Nachrichten, sollen mithilfe von *Complex Event Processing* (CEP) erhoben werden.

[1] Internetpräsentation der Forschungsgemeinschaft: `http://www.ADiWa.net/`
[2] Als ehemals eigenständiges Mitglied und jetziger Teil der Software AG.
[3] engl. Internet of Things (IoT)
[4] Radio-Frequency Identification (RFID). Eine Technologie zur Lokalisierung und Identifizieren von beliebigen Objekten.

Auf diese Weise ermittelte Daten werden als geschäftsrelevante *Ereignisse* bezeichnet. Das „Internet der Dinge“ kann nur dann voll ausgeschöpft werden, wenn die Verarbeitung und Zusammenfassung der zahlreichen verfügbaren Informationen zu geschäftsrelevanten Ereignissen automatisch und zeitnah geschieht. In diesem Kontext versucht ADiWa eine Möglichkeit zu schaffen, um die erfassten und gesammelten Ereignisse in aufbereiteter Form zu nutzen. Das Ziel ist es, laufende Geschäftsprozesse aufgrund von Ereignissen zu flexibilisieren, d. h. zu steuern und zu verändern. [ADi-2009b] Mit ADiWa soll die bestehende Lücke zwischen Datenerfassung und Datennutzung im Zusammenspiel mit IT-gestützten Geschäftsprozessen geschlossen werden.

Paket	*Bezeichnung und Beschreibung*
G1	Architektur des Gesamtsystems und Integration mit anderen Komponenten. Anforderungsanalyse und Koordinierung der ADiWa-Forschungsgemeinschaft
G2	Beschreibung, Bearbeitung und Erkennung komplexer Ereignisse. Aufarbeitung von Daten aus dem Internet der Dinge zu geschäftsrelevanten Ereignissen
G3	Güte, Analytik und Nutzung geschäftsrelevanter Ereignisse. Verwaltung von Qualitäts-Informationen der Ereignis-Verarbeitung
G4	Geschäftsprozess-Modellierung und -Simulation (Design-Zeit). Modellierung, Analyse und Simulation von Geschätsprozessen
G5	Geschäftsprozess-Ausführungscontrolling (Ausführungszeit). Ausführungsumgebung und Monitoring für Geschäftsprozesse, die auf geschäftsrelevante Ereignisse reagieren
G6	Prozess-Integration und Nutzer-Interaktion. Benutzer-Interface wird an den Geschäftszielen ausgerichtet.
G7	Sicherheitsmanagment dynamischer Geschäftsprozesse. Entwicklung des Sicherheitskonzepts für das gesamte System

Tabelle 1.1: Arbeitspakete in ADiWa

Das ADiWa-Forschungsprojekt ist in sieben Arbeitspakete eingeteilt. Die Tabelle 1.1 beschreibt diese Arbeitspakete. Das Arbeitspaket 5 „Geschäftsprozess-Ausführungscontrolling“ ist der Rahmen dieser Arbeit. [ADi-2010] In diesem Arbeitspaket wird untersucht, wie eine Ausführungsumgebung für flexible Geschäftsprozesse realisiert werden kann. Dazu wird eine dynamische Ausführung und Anpassung von Prozessen benötigt. Die Ausführungsumgebung für Geschäftsprozesse soll den Prozessverantwortlichen bei der flexiblen Ausführung unterstützen. Dafür muss sie es ermöglichen, auf prozessrelevante externe Ereignisse zu reagieren.[ADi-2010, Seite 54] Für dieses Arbeitspaket und für diese Arbeit ergeben sich aus dem ADiWa-Projekt zwei wesentliche Anforderungen: [ADi-2009a]

1. Geschäftsprozesse sollen dynamisch gesteuert und verändert werden. Diese Flexibilisierung basiert auf ausgewerteten Informationen der realen Welt.

2. Dazu müssen „Komplexe Ereignisse“, das heißt zusammengefasste Einzelereignisse, erkannt, verarbeitet und genutzt werden können.

Für die dynamische Steuerung und Änderung soll es möglich sein, dass Prozesse ad hoc an Abweichungen im Prozessablauf angepasst werden können. Damit sind Anpassungen gemeint, die auf im Prozessmodell nicht berücksichtigte Ereignisse reagieren. Unberücksichtigte Ereignisse können zu unwahrscheinliche, zu komplexe und zu seltene Ausnahmefälle sein, die so niemals modelliert werden würden. Beim Auftreten von unberücksichtigten Ereignissen soll das System zur Laufzeit in die Ausführung der Prozessinstanz eingreifen und sie verändern. So kann die Prozessinstanz sich an eine Situation anpassen, zu der sie durch die Modellierung nicht imstande gewesen wäre und erfolgreich beendet werden. [ADi-2010, Seite 36-37] Eine so angepasste Prozessinstanz wurde flexibilisiert. Welche Flexibilisierungen unterstützt werden und wie sie technisch realisiert werden können, dies zu beantworten ist Aufgabe dieser Arbeit.

1.1.1 Motivierendes Beispiel

Der folgende Beispielprozess ist einem tatsächlichen Geschäftsprozess eines international tätigen deutschen Logistikunternehmens entlehnt, das als Anwenderunternehmen Mitglied von ADiWa ist. Der Prozess bildet die Ankunft und Weiterverarbeitung von Sendungen am Frankfurter Flughafen ab. Abbildung 1.1 zeigt das BPMN-Modell dieses Prozesses.

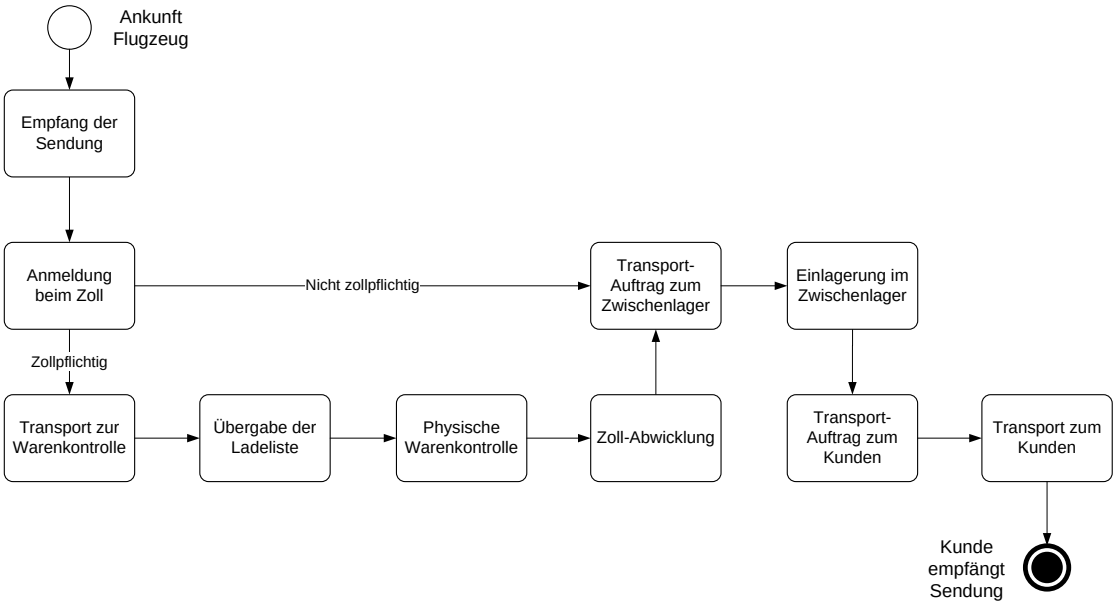

Abbildung 1.1: Beispielprozess „Import-Warenverteillager“

Nach der Ankunft des Transportflugzeuges wird die Sendung empfangen und beim Zoll angemeldet. Der Zoll überprüft, ob die Sendung zollpflichtig ist oder nicht. Ist sie es, wird

sie zur Warenkontrolle transportiert. Nachdem die Ladeliste mit detaillierten Informationen zur Ladung übergeben wurde, wird die Ware kontrolliert. Nach der Kontrolle werden ggf. Zölle erhoben und der Transport zum Zwischenlager in die Wege geleitet. Sollte die Sendung nicht zollpflichtig sein, wird die Sendung direkt von der Zoll-Anmeldung zum Zwischenlager-Transport aufgegeben. Nachdem die Ware im Zwischenlager angekommen ist, wird sie dort für den abschließenden Transport zum Kunden mit anderen Sendungen zusammengefasst und ausgeliefert. Dazu wird ein Transport-Auftrag aufgegeben und die Sendung transportiert. Der Prozess schließt mit dem Sendungs-Empfang durch den Kunden ab.

Dieses Beispiel zeigt das Prozessmodell zum Ablauf einer eintreffenden Sendung, wie sie in den meisten Fällen zutrifft. International operierende Logistikunternehmen müssen aber bei weltweiten Luftfrachtlieferungen mit unerwartet auftretenden Ereignissen rechnen und auf diese reagieren können. Dadurch werden flexiblere Prozesse notwendig.

Dynamische Flexibilisierung

Das obige Beispiel zeigt einen relativ unflexiblen Ablauf einer Luftfrachtsendung, die am Zielflughafen empfangen und zum Kunden weitergeleitet wird. Die einzige Flexibilität des Beispiels ist die Unterscheidung, ob eine Sendung zollpflichtig ist oder nicht. Diese statische Flexibilität ist im Prozessmodell dargestellt.

Das Anwenderunternehmen will als Neuerung seine Prozesse beim Auftritt von Ereignissen dynamisch flexibilisieren können. Ein solches Ereignis wäre beispielsweise die Verspätung des Flugzeuges. Flugverspätungen lassen sich in der Regel früh erkennen. Unmittelbar nach der Erkennung wird dem Speditionsunternehmen die drohende Verspätung mitgeteilt. Diese Mitteilung ist ein Ereignis. Sobald das Speditionsunternehmen über die Verspätung unterrichtet wurde, hat es einen gewissen Handlungsspielraum. Bei verspäteten Ankünften, sollen die Sendungen kritischer Kunden nicht erst im Zwischenlager eingelagert und zusammengefasst werden, sondern unverzüglich an den Kunden geliefert werden, notfalls als Einzel-Sendung. Dazu muss der Transport zum Kunden, sofern die Sendung nicht zollpflichtig ist, direkt nach der Zoll-Anmeldung in Auftrag gegeben werden. Ist die Sendung zollpflichtig, wird der Transport nach der Zoll-Abwicklung in Auftrag gegeben. Siehe dazu Abbildung 1.2. Im Prozessmodell wurden die Prozess-Aktivitäten „Transport-Auftrag zum Zwischenlager“ und „Einlagerung im Zwischenlager“ gelöscht. Auf diese Weise kann Zeit gewonnen werden, damit der Zeitverlust durch die Ankunft des Fluges keine Auswirkungen auf den Kunden hat.

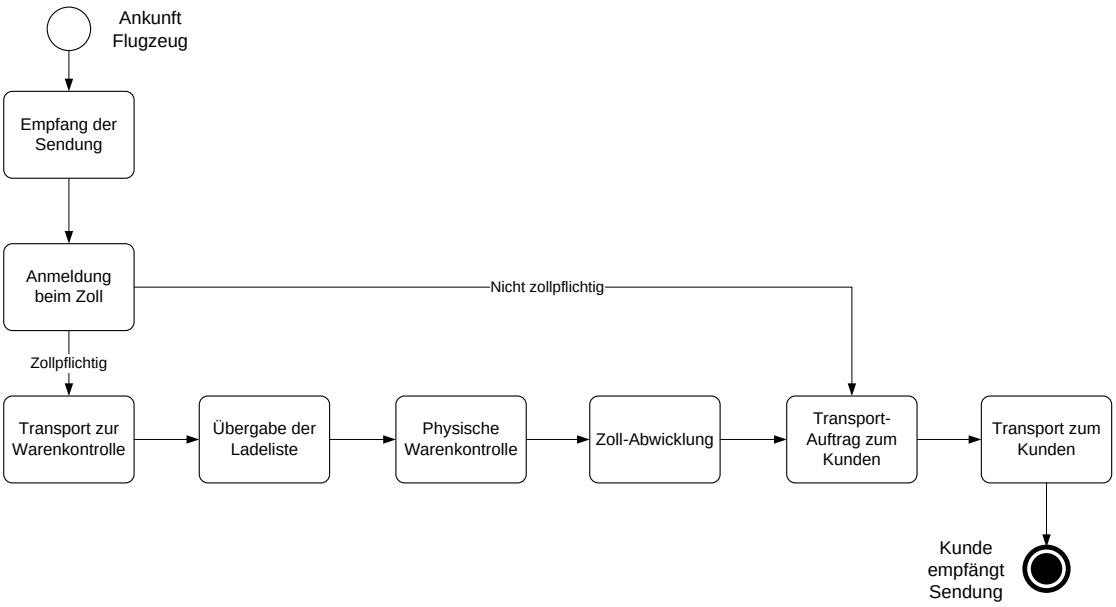

Abbildung 1.2: Flexibilisierter Beispielprozess „Import-Warenverteillager“

Auf diese bevorzugte Art können allerdings nicht alle Kunden behandelt werden. Die Vorzugsbehandlung soll nur Kunden zugestanden werden, die kritisch sind. Kritische Kunden sind z. B. Sendungsempfänger, die in der nahen Vergangenheit bereits von Verspätungen betroffen waren. Es sollen demnach nur die Sendungs-Prozesse durch die Vorzugsbehandlung flexibilisiert werden für die folgende Bedingung erfüllt ist:

Der Flug ist verspätet UND
Der Kunde war in den letzten drei Wochen bereits von mehr als zwei Verspätungen betroffen

Die Verspätung einer Sendung ist ein Ereignis. Zwei Verspätungen für einen Kunden ist innerhalb von drei Wochen ist dagegen ein *komplexes Ereignis*. Ein komplexes Ereignis birgt eine höherwertigere Information, die auf den einzelnen Verspätungs-Ereignissen der Vergangenheit basiert und viele Einzelinformationen zusammenfasst. Die Kriterien für ein komplexes Ereignis sind ebenso die Kriterien für die Prozessflexibilisierung, weil das komplexe Ereignis der Auslöser ist. Die Kriterien für eine Prozessflexibilisierung können beliebig erweitert und spezialisiert werden. Der Prozessablauf des Logistikunternehmens kann so ereignis-gesteuert flexibilisiert werden. Das hier genannte Beispiel ist nur eine von zahlreichen Möglichkeiten einen Prozess zu flexibilisieren. Welche komplexen Ereignisse eine Prozessflexibilisierung auslösen und an welcher Stelle und auf welche Art sie den Prozess flexibilisieren, liegt in der Hand des Prozessmodellierers.

1.1.2 Zentrale Fragestellung der Arbeit

An dem Beispiel sieht man die Notwendigkeit der ereignis-gesteuerten Flexibilisierung von IT-gestützten Geschäftsprozessen. Relativ statische Geschäftsprozesse sollen anhand von externen Ereignissen verändert werden können. Die Ereignisse können nicht Prozessmodell modelliert werden, da nicht jedes externe Ereignis für jeden Prozess auf die gleiche Weise behandelt werden kann. Im vorgestellten Beispiel muss z. B. jeder Kunde unterschiedlich behandelt werden. Außerdem können Ereignisse auftreten, die zum Zeitpunkt der Modellierung oder zum Prozessstart noch nicht bekannt sind.

Die ADiWa-Forschungsgemeinschaft will mithilfe der *webMethods Suite* der Software AG laufende Geschäftsprozesse anhand von auftretenden Ereignissen dynamisch verändern können. Damit sollen Probleme wie im Beispiel gelöst werden. Hierzu sollen, sobald ein bestimmtes komplexes Ereignis auftritt, Prozesse im Prozessausführungssystem automatisch flexibilisiert werden können. Um sich perspektivisch im CEP-Markt zu positionieren, entwickelt die Software AG dazu eine eigene CEP-Lösung (*webMethods Business Events*). Diese Lösung verfügt jedoch bisher über keine Anbindung zur eigenen Prozessausführungsumgebung. Um diese Anbindung umzusetzen, soll im Rahmen dieser Arbeit eine Schnittstelle entwickelt werden, die laufende Prozessinstanzen beim Auftritt eines Ereignisses dynamisch verändern kann. Mithilfe dieser Schnittstelle wird den Prozessverantwortlichen die Möglichkeit gegeben, auf nicht-modellierte Ereignisse außerhalb von Prozessen zu reagieren und den Prozess nach Bedarf zu flexibilisieren.

Die implementierte Schnittstelle soll später von einer CEP-Engine angesprochen werden, die komplexe Ereignisse aus vielen Basis-Ereignissen generiert. Dadurch können Prozesse an äußerliche Entwicklungen angepasst und jederzeit dahin gehend flexibilisiert werden. Die Prozesse laufen nicht mehr länger autark in ihrer Prozessausführung, sondern sind mit der Außenwelt verwoben und werden an neue Anforderungen ereignis-gesteuert angepasst. Auf diese Weise kann ein Unternehmen seine laufenden Prozesse schnell an neue Gegebenheiten anpassen. Durch die Flexibilisierung der Geschäftsprozesse wird auch das Unternehmen flexibler.

Als Fragestellung ergibt sich hieraus, wie Prozess-Flexibilität als Satz von frei wählbaren Funktionen bereitgestellt werden kann. Dazu muss zunächst bestimmt werden welche Arten der Geschäftsprozessflexibilisierung es gibt und wie sie sich in Flexibilitäts-Klassen gliedern und als Muster beschreiben lassen. Dazu sind zudem die theoretischen Grundlagen und die praktischen Möglichkeiten der Geschäftsprozessflexibilisierung zu untersuchen. Bei Bedarf sollen zusätzlich zu den bestehenden Mustern und Flexibilitäts-Klassen, eigene entwickelt werden. Für die ermittelten Muster sollen praktische Ansätze gefunden werden, die es ermöglichen sie innerhalb der webMethods-Suite zu implementieren. Die Ansätze sind soweit anzupassen und als Basis für weitere Entwicklungen zu

nutzen, dass sie die Muster möglichst vollständig implementieren. Die gefundenen und implementierten Ansätze sollen dann über eine Schnittstelle zwischen der Prozessausführung und webMethods Business Events bereitgestellt werden. Über diese Anbindung wird es einem Prozessmodellierer ermöglicht, eine frei wählbare Funktion zur Prozessflexibilisierung aufzurufen, die eine Menge von Prozessinstanzen an einer beliebigen Stelle beliebig flexibilisiert. Die Ansätze und die Schnittstelle sollen durch einen *Prozessmanipulator* implementiert werden, der die Menge von implementierten Ansätzen bereitstellt und von komplexen Ereignissen ausgelöst werden kann. Der Prozessmanipulator soll in der Lage sein, Prozessinstanzen unabhängig von ihrer Semantik und Geschäftslogik auf syntaktischer Ebene zu flexibilisieren.

1.2 Gliederung der Arbeit

Um Begriffe wie „Complex Event Processing“ und „Event-Driven Business Process Management“ vorzustellen, werden diese in Kapitel 2 eingeführt. Zur Vorstellung der verschiedenen Komponenten der webMethods Suite werden diese in Kapitel 3 beschrieben. Im folgenden Kapitel werden Flexibilitäts-Taxonomien untersucht und geeignete Klassen für die Auswahl und Bewertung von Flexibilitäts-Ansätzen identifiziert. Anschließend werden Flexibilitäts-Muster klassifiziert und vorgestellt. In Kapitel 5 werden die vorgestellten Muster dann auf ihre Anwendbarkeit im Kontext von webMethods untersucht und auf ihre Tauglichkeit hin überprüft. Das nachfolgende Kapitel 6 beschäftigt sich schließlich mit der Implementierung des Prototyps und seiner Vorstellung. Kapitel 7 schließt die Entwicklung mit der Integration der Ereignis-Quelle und dem Prozessmanipulator-Prototyp ab. Im letzten Kapitel 8 wird die Arbeit zusammengefasst und bewertet.

Kapitel 2

Einführung

2.1 IT-gestützte Geschäftsprozesse

IT-gestützte Geschäftsprozesse können als Prozessmodelle und Prozessinstanzen repräsentiert werden. Prozessinstanzen bilden dabei die Ableitung eines Prozessmodells. In dieser Arbeit werden, sofern nicht anders angedeutet, die Konzepte der *Business Process Modelling Notation* (BPMN) verwendet. [BPM-2011]

2.1.1 Prozessmodell

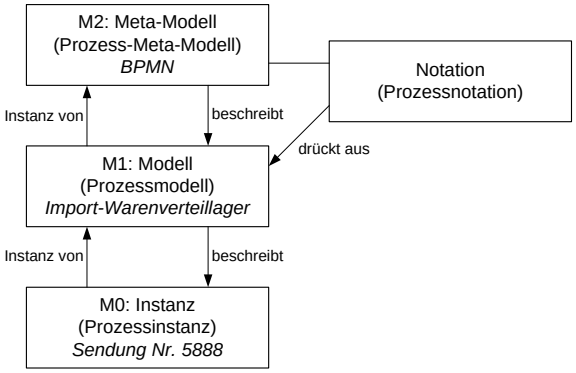

Abbildung 2.1: Geschäftsprozess in der Meta-Object Facility
Aufbauend auf [Wes-2007]

Ein IT-gestützter Geschäftsprozess wird in drei aufeinander aufbauenden Abstraktionsebenen dargestellt. Das Modell zur Modellierung des Prozessmodells ist die BPMN, die Notationen und Einschränkungen bereitstellt, um den Prozess zu modellieren. Von diesem Prozessmodell werden die Prozessinstanzen abgeleitet, die nach dem Abbild des Prozessmodells erstellt werden. Die Beziehung zwischen Prozess-Meta-Modell, Prozessmodell und Prozessinstanz kann mithilfe der Meta-Object Facility (MOF) ausgedrückt werden. Siehe Abbildung 2.1.

Ein BPMN-Prozessmodell besteht im wesentlichen aus vier Elementen[1], die im BPMN-Metamodell definiert werden. Das Prozessmodell selbst enthält eine Menge von Kanten und Knoten. Sequenzflüsse sind Verbinder, die die Beziehungen der Knoten zueinander ausdrücken und die Flussrichtung des Ablaufs definieren. Knoten können als Aktivität, Gateway oder Ereignis ausgeprägt sein.

Eine *Aktivität* repräsentiert eine Arbeitseinheit. Eine Arbeitseinheit kann z. B. ein Service-Task, Geschäftsregel-Task oder ein Benutzer-Task sein. Aktivitäten sind der funktionale Kern eines Prozessmodells. Sie repräsentieren die einzelnen Schritte, die notwendig sind, ein Prozessergebnis zu erreichen. *Gateways* werden zur Verzweigung und Zusammenführung von Sequenzflüssen genutzt. Dadurch können sie den Sequenzfluss des Prozesses leiten und steuern. Als Abweichung zur üblichen Definition (vgl. [Wes-2007]) sind Aktivitäten in dieser Arbeit[2] gleichzeitig Gateways. Die Aktivitäten können demnach über mehrere Ein- und Ausgänge verfügen und den Sequenzfluss zusammenführen und aufteilen. Üblicherweise benötigen Aktivitäten immer ein vor- oder nachgeschaltetes Gateway, um Sequenzflüsse zu teilen oder zu vereinen, da sie selbst keine Funktion zur Zusammenführung und Verzweigung haben.

Ereignisse im Verständnis der BPMN modellieren externe Ereignisse, die den Sequenzfluss des Prozesses verändern oder leiten können. Im einfachsten Fall repräsentieren sie den Anfang und Ende eines Prozesses. [Wes-2007] Gemäß der Spezifikation in der Version 2.1 unterstützt die BPMN Ereignisse im Prozessmodell. [BPM-2011] Neben den erwähnten Start- und Ende-Ereignissen sind das Zwischenereignisse, die etwa den Empfang einer E-Mail oder eine Zeitüberschreitung ausdrücken können. Ein Zwischenereignis kann für zwei unterschiedliche Fälle modelliert werden. Im ersten Fall wird ein Ereignis generiert, das für Teilnehmer außerhalb des Prozesses bestimmt ist. Zum Beispiel eine E-Mail, die versendet wird. Im zweiten Fall wird im Prozess auf ein externes Ereignis reagiert. Zwischenereignisse können im Gegensatz zu Start- und Ende-Ereignissen, die sich auf empfangen oder senden spezialisieren, immer empfangen und senden. [All-2009] Dabei handelt es sich aber nicht um ereignis-gesteuerte Flexibilität im Sinne dieser Arbeit. Nach diesem Verständnis sollen Ereignisse Prozesse frei wählbar verändern können. Modellierte Ereignisse sind hingegen starr und können nur das Verhalten auslösen, welches zuvor modelliert wurde.

Ein Sequenzfluss ist eine Kante, die zwei Elemente eines Prozessmodells miteinander verbindet. Ein Sequenzfluss wird dann aktiv, wenn die Sequenz vom Start-Ereignis übergeben wurde. Im Prozessablauf wird die Sequenz an Aktivitäten, Gateways oder Ereignisse übergeben und von diesen erneut an Sequenzflüsse weitergereicht. Sequenzflüsse

[1] Die Elemente „Swim-Lane“, „Daten-Assoziation“ und „Anmerkung“ werden nicht betrachtet

[2] Die Prozessausführung von webMethods kennt Aktivitäten, die gleichzeitig Gateways sind. Zu einem späteren Zeitpunkt wird dieses Feature genutzt.

können mit Bedingungen ausgestattet sein, die den Fluss nur bei erfüllter Bedingung ermöglichen. Ist die Bedingung nicht erfüllt, kann ein Standard-Sequenzfluss[3] definiert werden.

2.1.2 Prozessinstanz

Prozessinstanzen werden durch Prozessmodelle definiert. Instanzen existieren nur für einen kurzen, zeitlich eingeschränkten, Zeitraum. Alle Elemente von Prozessinstanzen sind Instanzen ihrer im Prozessmodell definierten Modelle. Das heißt, jede Aktivität in einer Instanz ist abgeleitet von der Aktivität im Prozessmodell.

Jede Prozessinstanz ist mit genau einem Prozessmodell assoziiert. Dabei kann jedes Prozessmodell beliebig oft abgeleitet sein. Analog gilt dies für Aktivitäts-, Ereignis- und Gateway-Instanzen und ihrer Modellierung im Prozessmodell. Da die Instanzen von ihren Modellen abgeleitet sind, werden Änderungen an Modellen auf alle Instanzen übernommen. Änderungen an Instanzen haben jedoch keine Bedeutung für das betreffende Modell.

Eine Instanz wird durch einen Prozesszustand beschrieben. Der Prozesszustand setzt sich zusammen aus den Daten der Instanz und dem Zustand des Prozessablaufs. Die Daten sind veränderlich und können sich von Instanz zu Instanz unterscheiden. Der Prozessablauf-Zustand beschreibt, an welcher Stelle die Instanz gerade durch den Sequenzfluss ausgeführt wird.

In dieser Arbeit werden die Begriffe „Prozess“ und „Prozessinstanz“ synonym verwendet. Unter Umständen wird eine genauere Abgrenzung zum Prozessmodell vorgenommen.

2.2 Geschäftsprozessflexibilisierung

Eine der treibenden Kräfte bei der Entwicklung von IT-gestützten Geschäftsprozessen war und ist der Wunsch nach mehr Flexibilität. In einer weniger technisierten Welt waren es nicht die Geschäftsprozesse, die flexibel waren, sondern die Menschen. Trat im Zusammenhang mit laufenden Prozessen ein Ereignis auf, lag es in der Hand eines höhergestellten Sachbearbeiters die zu leistende Arbeit anzupassen.

IT-gestützte Prozesse bieten die Möglichkeit, manuelle Tätigkeiten auszulagern und zu automatisieren. Im Prozess auftretende Arbeitsschritte können von vielen Bearbeitern gleichzeitig und dezentral bearbeitet und die zugehörigen Prozessinformationen jederzeit und überall abgerufen werden. [HCD+-1994] Am Anfang der Transformation der

[3] engl. Default

Geschäftsprozesse eines Unternehmens stehen einfache Abläufe, wie Gehaltsabrechnungen oder E-Mail-Benachrichtigungen bei Fehlern. Mit zunehmendem Einsatz werden aber auch komplexere Prozesse umgesetzt die einen zunehmenden Grad an Flexibilität verlangen. [DRRM-2010]

Die Prozesse müssen flexibler werden, da die Arbeit auf viele verteilte Einzelschritte aufgeteilt ist, die unterschiedlichen und zu anfangs nicht spezifizierten Bedingungen genügen müssen. Beispielsweise hängt der Versand von Sendungen an einem Warenverteillager von vielen zusätzlichen Faktoren ab, die auch nach der Ankunft der Sendung noch auftreten können. Zudem können einzelne Arbeitsschritte ausgegliedert und an Experten weitergereicht werden. Mit dem technischen Fortschritt und der zunehmenden Vernetzung verlassen IT-gestützte Geschäftsprozesse den Zugriffsbereich eines einzelnen Sachbearbeiters. Prozesse können tausendfach auftreten, komplex, unübersehbar und von vielen Orten und Ressourcen auf der Welt abhängig sein.

Flexibilität, so sagt die Wortdefinition, ist die Fähigkeit sich zu verändern, ohne die eigene Identität zu verlieren. Ein Geschäftsprozess ist dann flexibel, wenn es möglich ist, ihn zu verändern, ohne ihn auszutauschen. In einem *flexibilisierten* Prozess wurden also nur die Elemente verändert, die verändert werden mussten. Die unbetroffenen Elemente bleiben unverändert. [RW-2005][RSS-2006]

Geschäftsprozesse können aus verschiedenen Gründen flexibilisiert werden. Allen Gründen gemein ist der Wunsch, ein bestehendes Prozessmodell bzw. eine bestehende Prozessinstanz so anzupassen, dass er auf Anforderungen der Außenwelt reagieren kann. Nach Harrländer et al.[4] gibt es drei wesentliche Gründe für Prozessflexibilisierung [HSK-2004]:

- Prozessverbesserung: Ein bestehender Prozess soll marginal verbessert werden. Dadurch kann der Prozess effizienter werden. Der Prozess wird kontinuierlich und in kleinen Schritten verbessert. Das Risiko bei geplanten Änderungen ist moderat. Die Änderungen sind von kurzer Dauer und wirken permanent auf das Prozessmodell.
- Prozessinnovation: Der Prozess muss z. B. durch technologische Neuerungen oder gesetzliche Änderungen radikal verändert werden. Der Prozess wird einmalig und langwierig umgestaltet. Das Risiko einer solchen Änderung ist groß. Die Änderung betrifft einen großen Teil des Prozessmodells und ist geplant.
- Prozessadaption: Ein Prozess muss dann adaptiert werden, wenn sein Verlauf unter anderen Voraussetzungen geplant wurde, als denen die gegeben sind. Hier wird eine bestehende Prozessinstanz in unbestimmten Rahmen verändert. Die Änderung ist befristet und nicht geplant. Die Änderung erfolgt einmalig und ist von kurzer Dauer. Das Risiko und die Reichweite der Änderungen sind unbekannt.

[4] Mit Berufung auf Sadiq et al.

Die Prozessverbesserung und Prozessinnovation geht jeweils von Änderungen am Prozessmodell aus. Solche Prozessänderungen sind flexibel, weil sie auf weitreichende Änderungen in der geschäftlichen Umgebung reagieren und sich diesen anpassen. Bei der Prozessadaption werden hingegen nur bereits laufende Prozessinstanzen verändert. Thema dieser Arbeit ist die Prozessadaption laufender Prozessinstanzen. Die Prozessinstanzen sollen aufgrund von äußerlichen Ereignissen verändert werden können. Die Änderungen betreffen jedoch nur eine Menge von Instanzen und keinesfalls das gesamte Prozessmodell. Mit welchen Konzepten eine laufende Instanz adaptiert werden kann, wird im Laufe dieser Arbeit geklärt. Für die Prozessadaption existieren bereits mehrere Prozessflexibilisierungskonzepte, die im folgenden Abschnitt angesprochen werden.

2.2.1 Bestehende Prozessflexibilisierungskonzepte

An flexiblen Geschäftsprozessen wird seit längerem geforscht. In diesem Zusammenhang ist von flexiblen Workflow-Sprachen die Rede. Eine Workflow-Implementierung dieser Art ist *ADEPT*[5] bzw. *ADEPT2*[6], die seit etwa sieben Jahren an der Universität Ulm entwickelt werden (Stand Februar 2011). [DRRM-2010] Andere Vertreter sind *YAWL*[7], *FLOWer* jetzt *BPMOne*[8] oder *Declare*[9]. [SMR$^+$-2008] Flexible Workflow-Implementierungen sind durch ihr Design auf Flexibilität ausgerichtet. Beispielsweise überprüft ADEPT2 vor jeder Flexibilisierung einer Prozessinstanz, ob eine Flexibilisierung möglich ist und welche Auswirkungen sie auf die Prozessinstanz hätte. In ADEPT2 wird z. B. systemseitig überprüft, ob Instanzen bereits zu weit für eine Flexibilisierung fortgeschritten sind oder Flexibilisierungen mit der Modellierung zur Design-Zeit in Konflikt stehen. Dadam et al. zieht eine Grenze zwischen „gewöhnlichen“ Prozessausführungsumgebungen und flexiblen Workflow-Implementierungen, bei denen die herkömmlichen Systeme die Flexibilisierung zur Ausführungszeit nur bedingt, etwa durch Unterspezifizierung von Prozessteilen, und in minimalem Maße unterstützt und schnell die Robustheit der Prozessausführung gefährden. Flexibilisierungen dieser Art seien aufgrund ihres Risikos und geringen Unterstützung nur von Prozessspezialisten durchzuführen. [DRRM-2010]

Konzepte wie ADEPT2 sind jedoch proprietär. Sie bieten jeweils nur innerhalb ihrer Domänen eine Möglichkeit zur Geschäftsprozessflexibilisierung, weil sie an Workflow-Sprachen und Ausführungsumgebungen gekoppelt sind. Die bestehenden Workflow-Konzepte stehen für sich allein und lassen sich nicht in bereits etablierte Prozessausführungsumgebungen wie der von webMethods integrieren. Daher wird für webMethods in dieser

[5] `http://www.uni-ulm.de/in/iui-dbis/forschung/projekte/archiv/adept1.html`
[6] `http://www.uni-ulm.de/in/iui-dbis/forschung/projekte/adept2.html`
[7] `http://www.yawlfoundation.org/`
[8] `http://www.pallas-athena.com/home.html`
[9] `http://www.win.tue.nl/declare/`

Arbeit eine System-abhängige Prozessflexibilisierungs-Lösung gesucht, die sich an den Voraussetzungen und Besonderheiten des bestehenden Systems orientiert.

Innerhalb des ADiWa-Projekts gibt es neben dieser Arbeit eine weitere prototypische Implementierung, die im Rahmen der Dissertation von Markus Döhring in Zusammenarbeit mit SAP Research entwickelt wird. Der Prototyp führt automatisiert Änderungen am Sequenzfluss durch. Dazu werden Segmente zur Design-Zeit (unterspezifiziert) modelliert, die sich zur Laufzeit verändern können. Beim Betreten oder Verlassen eines solchen Segments werden Geschäftsregeln überprüft und eventuell Änderungen am Segment durchgeführt. Der Ablauf wird automatisch flexibilisiert und bedarf keiner externen Eingriffe. Der Döhring-Ansatz geht wie die vorliegende Arbeit von einer Flexibilisierung aus, die von Ereignissen ausgelöst wird, also ereignis-gesteuert ist. [DZG-2010] Die Abgrenzung zu der Lösung dieser Arbeit ist, dass diese Arbeit „Ad-hoc“-Änderungen ermöglicht soll, die nicht modelliert werden müssen. Ziel ist es, jedes Element eines Prozesses beliebig ändern zu können.

Die folgenden Abschnitte zeigen mit CEP und ED-BPM zwei Wege, einen Mehrwert aus geschäftlichen Ereignissen zu schaffen.

2.3 Complex Event Processing (CEP)

Ereignisse können von ganz unterschiedlicher Ausprägung sein und dabei technische Gegebenheiten wie Messungen der Temperatur, Systemereignisse wie Webseiten-Aufrufe oder Geschäftsereignisse wie Produktverkäufe oder Börsenkurse sein. [BD-2010] Einer Gartner-Untersuchung zufolge können Ereignisse auf verschiedene Weise im geschäftlichen Umfeld genutzt werden. [Sch-2011] Die einfachste Stufe der Ereignis-Nutzung ist die Ereignis-Weiterleitung. Bei der Weiterleitung werden Ereignisse von der Ereignis-Quelle ausgelesen und an eine Ereignis-Senke weitergeleitet. Auf die Weise können Ereignisse z. B. eine E-Mail-Benachrichtigung auslösen. Ähnlich funktioniert die Ereignis-Mediation der nächsten Stufe, die Ereignisse nicht nur weiterleitet, sondern umformt und in dieser Form an eine konforme Ereignis-Senke weiterleitet. Die Mediation ist bei heterogenen Ereignis-Quellen notwendig, die Ereignisse in nicht zu verarbeitender Form generieren.

Complex Event Processing (CEP) ist die nächste Stufe. Mithilfe von CEP werden die Ereignis-Ströme vieler Ereignis-Quellen gefiltert und zu komplexen Ereignissen aggregiert. CEP ist ein Konzept, das Ereignisse definiert, analysiert und weiterverarbeitet. [AEE$^+$-2009b] Das Ergebnis dieser Weiterverarbeitung sind komplexe Ereignisse. Komplexe Ereignisse enthalten in abstrahierter Form höherwertigere Informationen als ihre zugrunde liegenden Ereignisse (siehe Abbildung 2.2). Komplexe Ereignisse können wesentlich zu geschäftlichen Entscheidungen beitragen, indem sie Trends und Muster der

Außenwelt anzeigen und Unternehmensarchitekturen agiler, reaktionsschneller und echtzeitfähig machen. [BD-2010][CS-2009]

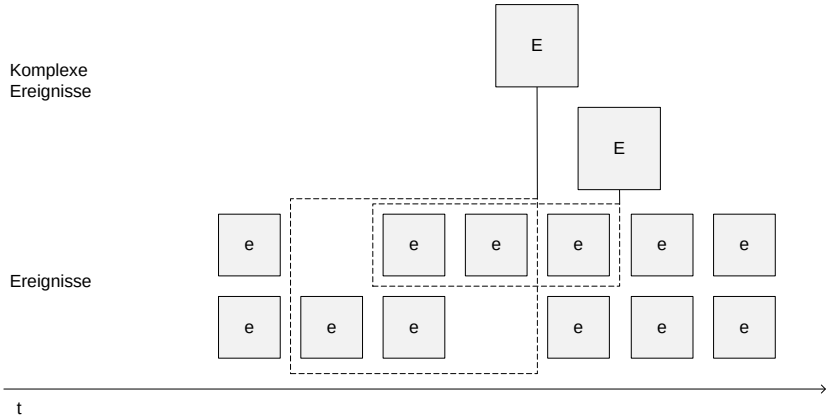

Abbildung 2.2: Ereignisse und komplexe Ereignisse

Die ankommenden Ereignisse zu einer CEP-Engine liegen in elektronischer und interpretierbarer Form vor. Diese Ereignisse können über Schnittstellen abgefragt werden, die Schnittstellen durchaus heterogen sein können. Mithilfe von Adaptern werden die eingehenden heterogenen Ereignisse in standardisierte Ereignisse umgewandelt. Der stete Fluss von Ereignissen summiert sich zu Ereignisströmen, die von Ereignisquellen ausgehen. Komplexe Ereignisse summieren sich analog zu komplexen Ereignisströmen, die ihrerseits auch wieder als Eingangs-Strom der CEP-Engine dienen können.

Mithilfe von CEP können z. B. die vielen Sendungsverspätungs-Ereignisse eines Kunden als komplexes Ereignis ausgewertet werden. Damit kann ein komplexes Ereignis generiert werden, das zusammenfasst, dass ein Kunde in der Vergangenheit bereits mit Verspätungen umgehen musste. Dazu muss eine Abfrage (Query) auf den steten Strom an Ereignissen gesetzt sein, die die Ereignisse überprüft und bei der Gültigkeit der Abfrage ein komplexes Ereignis generiert. Die folgende Tabelle 2.1 zeigt einen Ereignisstrom s und einen daraus gebildeten komplexen Ereignisstrom S mithilfe der Abfrage „Generiere komplexes Ereignis, wenn die Anzahl von Verspätungen in s für einen bestimmten Kunden gleich drei ist“.

Zeit	*Ereignisstrom s*	*Komplexer Ereignisstrom S*
t	S1 für Kunde K1 verspätet	...
t+1	S2 für Kunde K2 verspätet	...
t+2	S3 für Kunde K3 verspätet	...
t+3	S4 für Kunde K1 verspätet	...
t+4	S5 für Kunde K1 verspätet	„*Kunde K1 von drei Verspätungen betroffen*“
t+5	S6 für Kunde K4 verspätet	...

Tabelle 2.1: Ereignisstrom und komplexer Ereignisstrom

Der Grundgedanke von CEP ist keineswegs neu. Im geschäftlichen Umfeld werden schon immer Ereignisse aus verschiedenen Quellen ausgewählt, analysiert und weiterverarbeitet. Um relevante zukünftige Trends und Entwicklungen zu erkennen, lesen Angestellte beispielsweise die Zeitung, betrachten Börsenkurse oder verfolgen die Preisentwicklung von Rohstoffen. Auf Basis der gewonnenen Informationen versuchen sie einen geschäftsrelevanten Zusammenhang zu ermitteln, und diesen auf ihr zukünftiges Handeln anzuwenden.

CEP automatisiert diesen Vorgang. Eine CEP-Engine betrachtet die ankommenden Ereignisse und summiert sie, aggregiert sie, wählt sie aus oder verwirft sie. Dabei findet eine Auswahl nach zeitlichen Abständen und Ereignis-Eigenschaften statt. Eine CEP-Engine kann Muster (engl. „Pattern“) von Ereignissen in Ereignisströmen finden, z. B. „Ereignis A wird gefolgt von Ereignis B, anschließend folgt Ereignis C oder D“. Darüber hinaus kann zeitlich und räumlich abgegrenzt werden. Etwa mit den Einschränkungen „innerhalb der letzten zwei Minuten“ oder „innerhalb eines Radius von zwei Kilometern“.

Mit CEP können reale Ereignisse unterschiedlicher Ereignisströme betrachtet und zueinander in Verbindung gebracht werden. Eine Häufung von Übernahme-Gerüchten in den Nachrichten und der stetige Aufkauf von bestimmten Wertpapieren eines Unternehmens, löst z. B. das komplexe Ereignis „Firmenübernahme steht an“ aus. Ein komplexes Ereignis ist also die Aggregation einer Menge von realen Ereignissen. Durch diese Auswahl ist ein komplexes Ereignis eine abstrahierte Darstellung einer Menge relevanter Basis-Ereignisse aus dem tatsächlichen geschäftlichen Umfeld. Die Auswahl relevanter Ereignisse bedeutet auch, dass irrelevante Ereignisse vernachlässigt werden. Irrelevante Ereignisse treten durch die riesige Anzahl von Ereignissen naturgemäß in großer Zahl auf.

Die Abfrage für die Auswahl einer Menge von relevanten Ereignissen wird in einer *Event Processing Language* (EPL) beschrieben. [AEE+-2009b] Etablierte Produkte wie StreamBase[10], Sybase Aleri Studio[11] oder Progress Apama[12] und auch webMethods Business Events nutzen dazu eine Mischung aus grafischer Modellierung und SQL-ähnlichen Abfragesprachen [For-2009].

[10] `http://www.streambase.com`
[11] `http://www.sybase.com/`
[12] `http://web.progress.com/en/apama/`

2.4 Event-Driven Business Process Management (ED-BPM)

Nach Gartner ist die höchste Stufe der Ereignis-Verarbeitung das *Event Driven Business Process Management* (ED-BPM). Mit ED-BPM werden Ereignisse genutzt, um Geschäftsprozesse zu steuern und zu verändern. Der Begriff des Event Driven Business Process Management wurde maßgeblich durch Rainer von Ammon et al. geprägt. [AESW-2008][AEE$^+$-2009a][AEE$^+$-2009b] Erstmals erwähnt wurde ED-BPM noch als „Event-based BPM“ durch Roy Schulte im Jahr 2006. [Etz-2010]

ED-BPM erweitert IT-gestützte Geschäftsprozesse so, dass sie durch den Fluss von Ereignissen gesteuert werden können. Tritt z. B. ein Prozess-relevantes Ereignis auf, muss der Prozess so umgestaltet werden, dass er die veränderten Bedingungen berücksichtigt und trotzdem erfolgreich beendet werden kann.

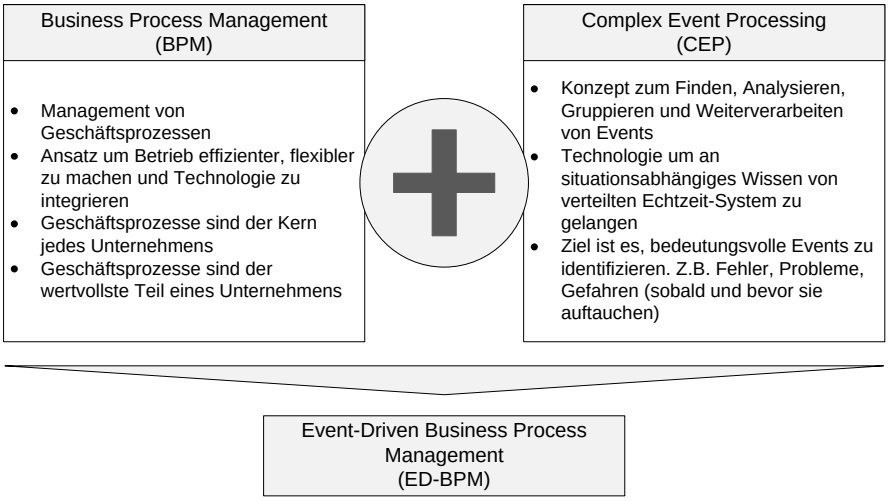

Abbildung 2.3: EDPM als Kombination von BPM und CEP [AEE$^+$-2009a]

Das ED-BPM bildet die Verbindung von CEP, *Event-Driven Architectures* (EDA) und *Business Process Management* (BPM). [BD-2010] Durch die Verbindung können die durch CEP aggregierten Ereignisse direkt in der Prozessausführungsumgebung auf Prozesse angewandt werden. Prozesse der Ausführungsumgebung können mithilfe der komplexen Ereignisse gesteuert und verändert werden. Siehe dazu Abbildung 2.3.

Event-Driven BPM ist eine Teildisziplin von Event-Driven Architectures. EDA sind komplementär zu konventionellen Software-Architekturen und Service-orientierten Architekturen (SOA). Die Autoren Bruns und Dunkel sehen eine EDA erst durch CEP und die Ereignis-Verarbeitung als „richtig“ ereignis-gesteuert an. [BD-2010]. Wie in einer EDA werden beim ereignis-gesteuerten BPM Prozesse nicht mehr nur synchron gesteuert, sondern asynchron durch komplexe Ereignisse. Diese Ereignisse bewirken ein bestimmtes

Prozessverhalten. Idealerweise werden im Prozessverlauf und nach Prozessende zusätzliche Ereignisse generiert, die den Zustand des Systems beschreiben. [Luc-2007] Jeder Prozess kann also über Ereignisse gesteuert werden und seinen Prozesszustand über Ereignisse ausdrücken. Zum Beispiel steuert das Ereignis „Stoppe Prozess A“ den Prozess, während das Ereignis „Prozess gestoppt“ den daraus folgenden Zustand beschreibt.

Ob ein Ereignis für einen Prozess relevant ist oder nicht, wird dadurch festgestellt, dass Prozessinformationen und Ereignisse miteinander korreliert werden. Ob eine Prozessinstanz, ein Prozessmodell oder eine Menge von Instanzen betroffen ist oder nicht, entscheidet die Geschäftslogik. Laut von Ammon wird durch ED-BPM neben dem Prozessmodellierer auch ein Ereignis-Modellierer notwendig. [AEE$^+$-2009b] Der Ereignis-Modellierer wählt oder implementiert eine geeignete Verhaltensweise, sobald ein externes Ereignis eintritt.

Der Vorteil des ED-BPM, nach Ammon, gegenüber dem herkömmlichen BPM besteht darin, dass ein Geschäftsprozess zu jedem Zeitpunkt nicht nur Abhängig vom eigenen Zustand ist, sondern auch externe Ereignisse und den Zustand des restlichen Systems den Prozess beeinflussen. Durch diese zusätzlichen Informationen kann der Prozess auf interne und externe Veränderungen flexibel reagieren bzw. „reagiert werden“. Zusätzlich übertrifft ED-BPM die beschriebenen BPMN-Ereignisse dadurch, dass auch externe Ereignisse außerhalb des modellierten Prozesses den Ablauf eines Prozesses beeinflussen können. Während auftretende komplexe Ereignisse den gesamten Ablauf, Aufbau und Ausführung eines Prozesses flexibilisieren können, kann durch Ereignisse der BPMN der Prozess nur eingeschränkt flexibilisiert werden. Diese Flexibilisierungen müssen allerdings bereits im Prozess modelliert sein.

Dadurch, dass Event-Driven BPM Ereignisse in Echtzeit generiert, kann durch *Business Activity Monitoring* (BAM) der Prozess in Echtzeit überwacht werden. Mithilfe von BAM sind Unternehmen in der Lage, den Informationsgehalt von Ereignissen und Prozessen zu verbinden. So können Prozesse erkannt werden, die vom beabsichtigten Prozessablauf abweichen und verändert werden sollten. [Ora-2010]

Aufgrund des frühen Entwicklungsstadiums von ED-BPM fehlen jedoch allgemein akzeptierte Standards und Best-Practices. Auch die zentralen Bereiche wie eine Event Processing Language (EPL), die komplexen Ereignisse (Ereignismodelle) und Schnittstellen der CEP-Engine mit der Umwelt, wie z. B. der Prozessausführungsumgebung, sind nicht standardisiert. Zumeist werden dazu proprietäre Lösungen implementiert. Darüber hinaus verfügt man in der Wirtschaft noch über wenig Erfahrung im Umgang mit ED-BPM. Die Entwicklung von CEP-Anwendungen hat gerade erst begonnen und wurde in den letzten Jahren zumeist durch akademische Entwicklungen in Universitäten und Hochschulen getrieben. [BD-2010, Seite 224]

Kapitel 3

Ereignis-gesteuertes BPM und webMethods

Dieses Kapitel definiert die konzeptionelle Anbindung von der webMethods CEP-Engine an die Prozessausführung im Sinne von ED-BPM. Mithilfe dieser Anbindung können Prozessinstanzen durch externe Ereignisse gesteuert oder verändert werden. Dazu werden in diesem Abschnitt die webMethods-Komponenten vorgestellt, die für die Anbindung relevant sind.

Die Abbildung 3.1 auf Seite 19 zeigt, an welcher Stelle im ADiWa-Projekt der Prozessmanipulator implementiert werden soll (Kreis mit Markierung). Die massiv umrandeten Elemente „CEP“, „BPMS“ (Prozessausführung) und „Event-Bus“ (Ereignis-Bus) sind Thema der Arbeit und werden zur Realisierung eingebunden. In der Abbildung 3.2 auf Seite 20 werden die relevanten Komponenten in die Architektur eines ereignis-gesteuerten Systems für die webMethods-Suite (WM) eingebunden. Die Abbildung zeigt, wie heterogene Ereignisse auf dem Ereignis-Bus ausgegeben werden. Die CEP-Engine korreliert die Ereignisse (gestrichelte Linien) zu neuen komplexen Ereignissen (durchgezogene Linie). Tritt eine bestimmte Ereignis-Folge auf die eine Ereignis-Abfrage zutrifft, löst die CEP-Engine das Ereignis „Prozessmanipulation“ („E“) aus. Dieses Ereignis beschreibt die Flexibilisierung und trägt als Problemlösung eine Anweisung an den „Prozessmanipulator“ mit sich. Beispielsweise könnte das die Anweisung sein, eine Menge von Prozessinstanzen zu stoppen.

Die einzelnen Komponenten dieser Architektur sind Design-Entscheidungen der Software AG. Die vorliegende Arbeit bietet eine konzeptionelle Ausarbeitung und prototypische Implementierung des „Prozessmanipulators“ als funktionaler Kern dieser Architektur. In den folgenden Abschnitten werden die für die Flexibilisierungs-Ansätze und den Prototyp benötigten Komponenten der webMethods Product-Suite vorgestellt.

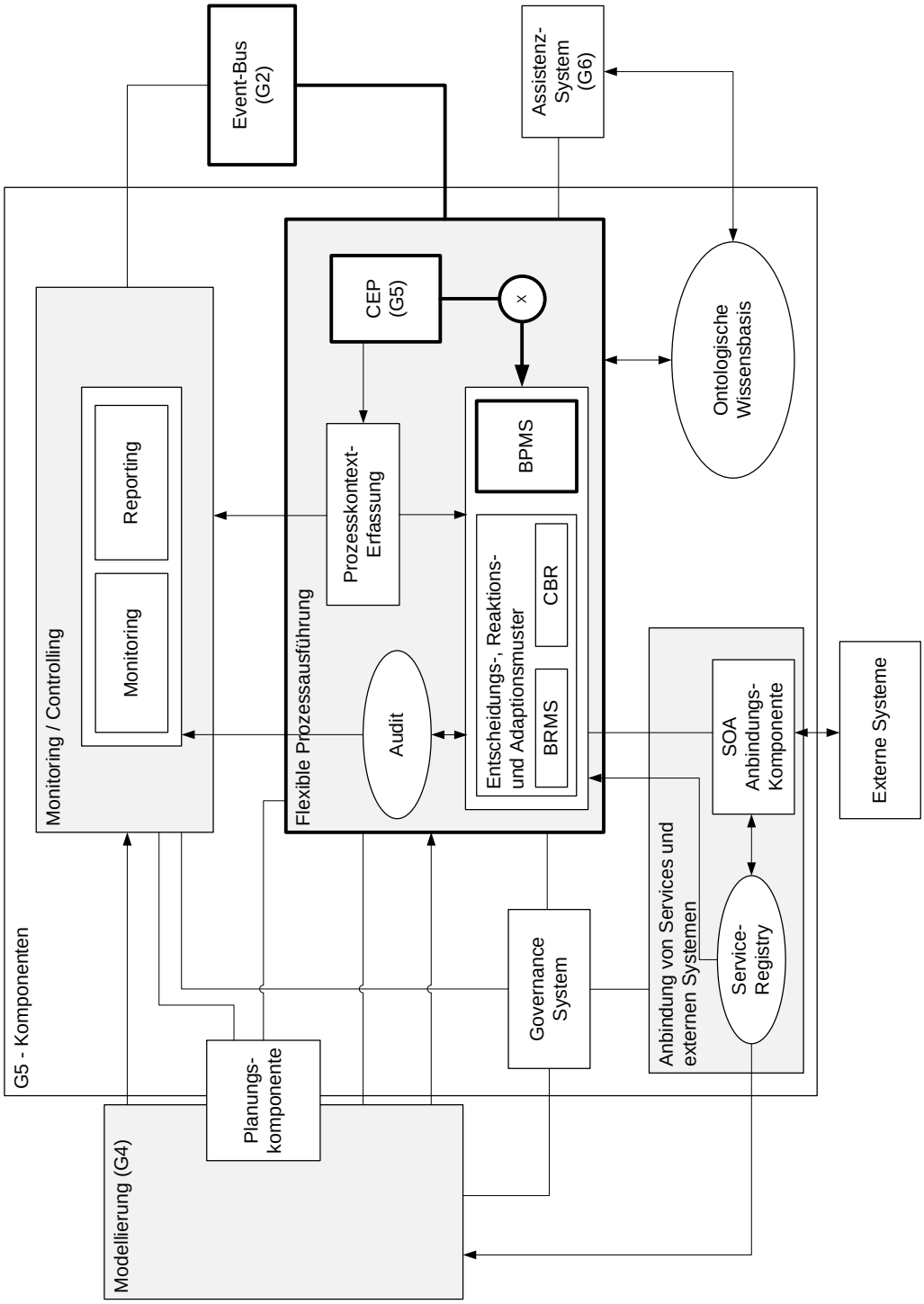

Abbildung 3.1: Das Thema der Arbeit im Kontext von ADiWa
[Internes und nicht veröffentlichtes Dokument der SAG]

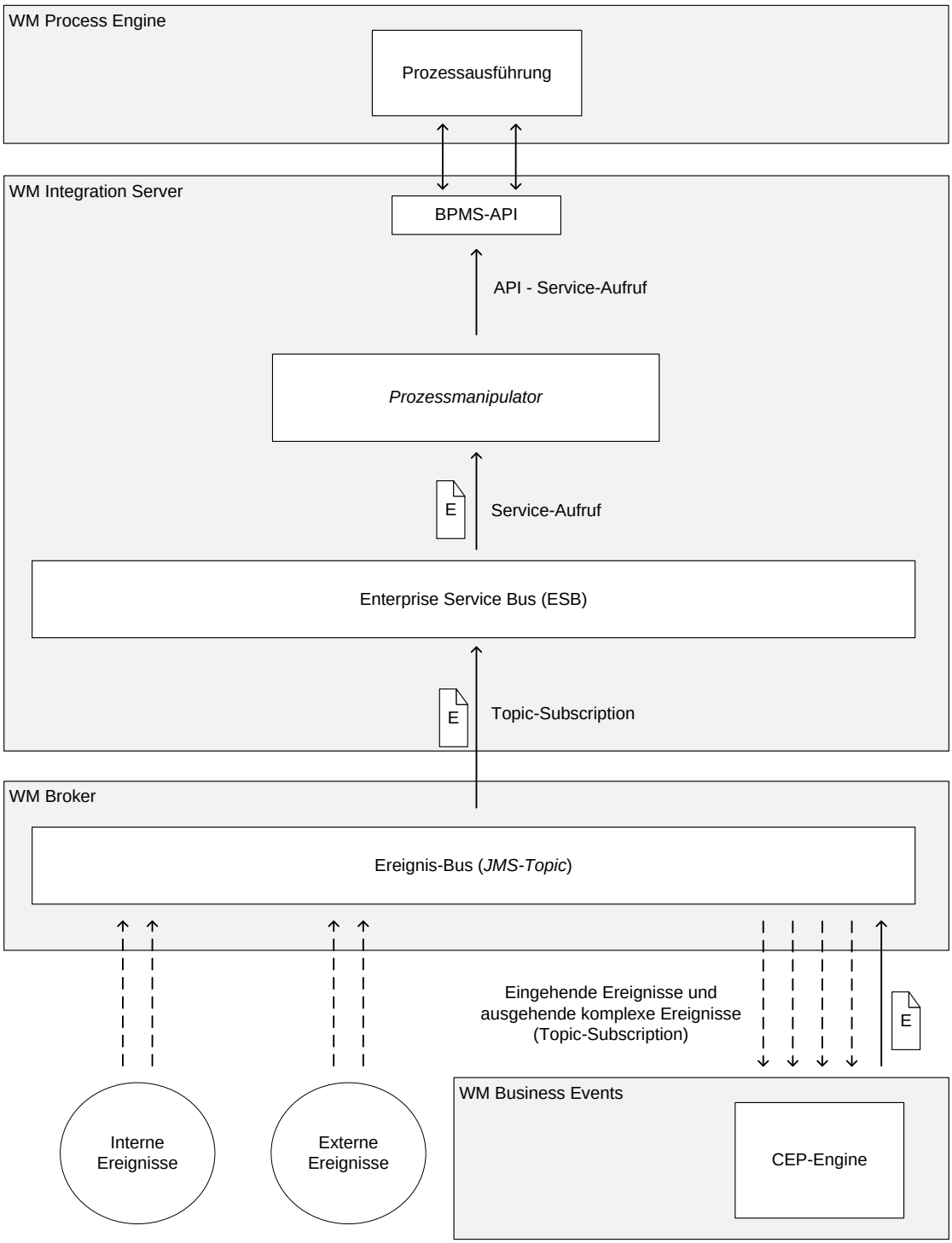

Abbildung 3.2: Anbindung von CEP und BPMS (ED-BPM)

3.1 Die webMethods Product Suite

Die webMethods Product Suite ist ein Softwarepaket der Software AG das Zuvor war webMethods ein eigenständiges amerikanisches Unternehmen mit Sitz in Reston. [Sof-2010]

Die folgenden Komponenten sind für die Entwicklung des Prototyps von Bedeutung und werden im Hinblick auf ihr Mitwirken beschrieben. Die Suite besteht neben den detailliert vorgestellten Komponenten u. a. noch aus den Produkten CentraSite, Optimize, EntireX und ApplinX, die für den Prototyp keine Bedeutung haben. [Sof-2009c]

3.1.1 webMethods Business Events

webMethods Business Events ist die in der Entwicklung befindliche CEP-Engine von webMethods. Business Events korreliert Ereignisse zu komplexen Ereignissen. Dazu werden Ereignisse vom Ereignis-Bus gelesen, komplexe Ereignisse generiert und diese Ereignisse wiederum auf dem Ereignis-Bus ausgegeben. Die CEP-Engine basiert auf dem „RTM Analyzer“ der Realtime Monitoring GmbH (RTM) einem Spin-off der Universität Marburg. Nachdem zuvor die Produkte namhafter Hersteller für CEP-Lösungen wie Apama oder Tibco, auch im Rahmen dieser Arbeit, evaluiert wurden, wurde RTM als CEP-Engine für webMethods ausgewählt. Unmittelbar danach, im April 2010, kaufte die Software AG RTM. [Bus-2010]

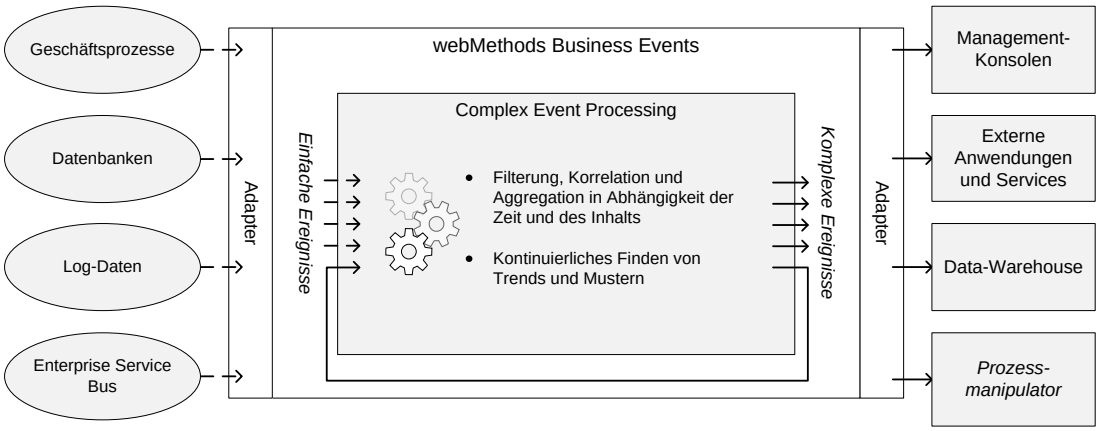

Abbildung 3.3: Funktionsweise der CEP-Engine
Aufbauend auf [Rea-2010]

Die Generierung von komplexen Ereignissen mit Business Events (vormals RTM Analyzer) besteht aus drei Schritten. Zuerst werden mit bereitgestellten Eingangs-Adaptern, beliebige Ereignis-Quellen abgefragt. Über eine SQL-verwandte Abfrage-Sprache wird

anschließend die Geschäftslogik, d. h. die Ereignis-Abfrage (Query) modelliert. Die Query wird durch die CEP-Engine fortlaufend und auf die Menge aller eingehenden Ereignisse ausgeführt. Zuletzt sorgt ein Ausgangs-Adapter dafür, dass die komplexen Ereignisse an Ereignis-Senken gesendet werden können. Zum Beispiel für das Monitoring oder die Weiterleitung der komplexen Ereignisse. Siehe Abbildung 3.3.

In dieser Arbeit werden alle eingehenden Ereignisse mithilfe des webMethods Brokers zur CEP-Engine gesendet. Nach der Aggregation in der CEP-Engine werden die komplexen Daten ihrerseits über den Broker als Eingangsdaten zum Prozessmanipulator gesendet. [Rea-2010]

3.1.2 webMethods Broker

Der *webMethods Broker* ist ein Nachrichten-Router, der asynchronen und synchronen Nachrichtenfluss ermöglicht. Der Broker funktioniert als Vermittler, der Daten vom Sender zum Empfänger weiterleitet. Normalerweise werden Nachrichten innerhalb der webMethods Suite nur durch den Integration Server gesendet und empfangen. Anhand des *Java Message Services* (JMS) und einer proprietären Broker-API können jedoch zusätzliche Sender und Empfänger vom Bus lesen und auf ihn schreiben. Auf diese Weise kommuniziert Business Events mit dem Broker.

Der Broker stellt mit dem *Ereignis-Bus* den Nachrichten-Bus für Ereignisse zur Verfügung. Der Ereignis-Bus selbst ist als ein JMS–Topic implementiert. Der Java Message Service unterscheidet zwischen den beiden Verbindungs-Typen *Topic* für Publish-Subcribe-Kommunikation und *Queue* für Punkt-zu-Punkt-Verbindungen. Beliebig viele Komponenten können sich also als Subscriber beim JMS-Topic eintragen lassen und werden dann mit allen Ereignissen versorgt, die auf dem Ereignis-Bus liegen. Während sich die CEP-Engine am Ereignis-Bus abonniert, um interne und externe Ereignisse zu empfangen und um komplexe Ereignisse zu senden, abonniert sich der Prozessmanipulator ausschließlich als Empfänger dieser Ereignisse.

Das Abonnement auf Ereignisse wird über einen Trigger im *Integration Server* (IS) (siehe folgender Abschnitt) definiert. Dieser Trigger bestimmt das Verhalten für Ereignisse, die auf einem JMS-Topic liegen. So kann konfiguriert werden, dass der Prozessmanipulator beim Auftreten von Prozessmanipulations-Ereignissen gestartet wird.

Die über den Broker versendeten und empfangenen Nachrichten werden als *Dokumente* repräsentiert. Ein Dokument ist die Instanz eines *Dokument-Typs*, der den Aufbau der Nachricht in einer Schema-ähnlichen Notation beschreibt. Ein Dokument kann beliebig

viele Daten in Form von Strings, Integern, Listen etc. enthalten. [Sof-2009c] Jedes komplexe Ereignis auf dem Ereignis-Bus ist als ein Dokument modelliert. In diesem Fall handelt es sich dabei um den Dokument-Typ „Prozessmanipulation“.

3.1.3 webMethods Integration Server

Der Integration Server (IS) ist die Ausführungsumgebung für Services und dient als *Enterprise Service Bus* (ESB). Ein ESB ist ein gemeinsam genutzter Kommunikations-Bus zur Integration von heterogenen Systemen bzw. von Service-orientierten Architekturen. [Cha-2004] Der IS empfängt Anfragen von Clients und authentifiziert und autorisiert sie. Anschließend werden die gewünschten Services ausgeführt und die Eingangsdaten als Parameter übergeben. Nach der erfolgreichen Ausführung werden die Rückgabe-Daten des Services an den Client zurückgegeben. [Sof-2009c][Sof-2009a]

Im Integration Server werden sogenannte „Built-In-Services“ (API-Services) angeboten. Diese Services können als eine API zum IS und anderen webMethods-Komponenten verstanden werden. Zur Gliederung der verschiedenen API-Services sind diese in funktionale Pakete gegliedert. Zusätzlich zu den bestehenden Services können eigene Pakete und Services erstellt werden. Auch der Prozessmanipulator soll als Service implementiert werden. Wie im vorherigen Abschnitt erwähnt wurde, wird der Prozessmanipulator immer dann gestartet, wenn ein Prozessmanipulations-Ereignis auftritt. Die Prozessmanipulation findet also ereignis-gesteuert statt.

3.1.4 webMethods Process Engine

Die Process Engine ist die Prozessausführungsumgebung der webMethods-Suite. Die Engine ist als Paket von Services zur Prozesssteuerung im Integration Server implementiert. [Sof-2009b] Aufgabe der Process Engine ist es, Instanzen von Prozessmodellen zu steuern, und deren Einzelschritte aufzurufen. Dazu können je Einzelschritt z. B. externe oder interne Services gestartet oder menschliche Aktivitäten angefordert werden.

Bereits laufende Instanzen können mit *webMethods Monitor* überwacht werden. Monitor bietet die Möglichkeit den Zustand von Prozessinstanzen und ihrer Einzelschritte zu überwachen und Prozessinstanzen zu pausieren, stoppen, wiederaufzunehmen oder zu wiederholen. [Sof-2009b] Für die Steuerung und Überwachung von Prozessen dient die Web-basierte Administrations- und Benutzer-Konsole „*My webMethods*“, die auf webMethods Monitor aufbaut. Neben der Administration der webMethods Suite bietet sie eine grafische Darstellung laufender Prozessinstanzen und existierender Prozessmodelle.

Prozesse in webMethods

Prozesse in webMethods werden im Software AG Designer modelliert. Ein dort erstelltes Prozessmodell liegt in einer BPMN-ähnlichen Notation vor. Das Modell selbst wird in einem proprietären XML-Dialekt repräsentiert. Über dieses XML-Dokument kann auf die einzelnen Elemente des Modells zugegriffen werden.

Die Knoten des Prozessmodells in webMethods werden als Ausprägung eines *Steps* modelliert. Solche Ausprägungen sind z. B. Invoke-Steps für Aktivitäten, Receive-Steps für Start-Ereignisse, Terminate-Steps für Ende-Ereignisse oder Gateway-Steps für Gateways. Aktivitäten, Ereignisse und Gateways werden im Folgenden als Steps bezeichnet, da sämtliche Steps ähnlich modelliert sind. Über die XML-Abstraktion kann unabhängig von ihrer Funktionalität transparent auf sie zugegriffen werden. Ein Prozess muss mindestens ein Eingangs-Ereignis und ein Ende-Ereignis haben. Jedem Receive-Step kann ein Dokument-Typ zugewiesen werden, bei dessen auftreten eine Prozessinstanz gebildet wird. Bei den Terminate-Steps kann ausgewählt werden, welchen Zustand die Prozessinstanz annimmt, wenn sie den Step erreicht hat. Gewählt werden kann zwischen „Abgebrochen“ und „Erfolgreich abgeschlossen“. Welcher Art von Aktivität durch einen Invoke-Step ausgeführt wird, wird in den Eigenschaften der Aktivität definiert. Der Prozessmodellierer kann für Service-Tasks zwischen Web-Services und IS-Services auswählen. Alternativ dazu kann eine Aktivität als eine menschliche Aktivität oder Business-Task-Rule modelliert werden.

Die Prozessausführung kann ein Prozessmodell nach dem Aufspielen in Versionen ablegen. Für jeden Invoke-Step wird anschließend ein eigener Trigger angelegt, der das Verhalten beim erreichen der Aktivität durch den Sequenzfluss auslöst. Prozess-Ablauf in der Process Engine und Prozess-Überwachung im Monitor laufen getrennt voneinander ab.

3.1.5 Software AG Designer

Der Software AG Designer ist das zentrale Entwicklungstool der webMethods-Suite. Designer ist ein Eclipse-basiertes Tool u. a. zur Entwicklung von IS-Services, dem Entwurf von CEP-Querys und der Modellierung von Geschäftsprozessen.

Zur Service-Entwicklung können Java-Services implementiert werden, die im Integration Server persistent abgelegt werden. Wird ein Service ausgewählt, wird der Source-Code des Services in einem Java-Editor angezeigt und der Service mit den von Eclipse gewohnten Features bearbeitet. Sobald der Service gespeichert wird, wird er in den Integration Server geladen und ist, sofern fehlerfrei, lauffähig.

Zur Modellierung von CEP-Abfragen wird die in der Entwicklung befindliche CEP-Modellierungs-Sicht verwendet, die eine grafische und textuelle Modellierung anbietet. Die Query-Entwicklung erfordert einen oder mehrere eingehende Ereignis-Ströme und grafisch modellierte komplexe Ereignisse. Die Ereignis-Abfragen werden in SQL definiert und auf ihre Korrektheit überprüft. Die resultierende Query wird mitsamt der Ereignis-Definitionen in die CEP-Engine aufgespielt.

Bei der Prozessmodellierung wird ein BPMN-ähnliche Notation verwendet, das die meisten der BPMN-Elemente abbildet. Nach der Modellierung können z. B. für die Implementierung der Prozess-Aktivitäten IS-Services oder Web-Services ausgewählt werden. Bevor Instanzen des Modells gestartet werden können, wird das Modell zunächst geprüft und in die Prozessausführung „Process Engine“ aufgespielt. Mithilfe der Debug-Perspektive kann das aufgespielte Modell anschließend zudem debuggt werden.

3.2 Prozessmanipulator

Der Prozessmanipulator ist der Prototyp, der in dieser Arbeit entwickelt wird, und soll als ein Java-Service im Integration Server implementiert werden. Wenn ein komplexes Ereignis eintritt, wird das Ereignis in ein Dokument umgewandelt, das dem Service als Parameter übergeben wird. Aus diesem Dokument zieht der Prozessmanipulator seine Informationen und führt die darin definierte Flexibilisierung von Instanzen aus. Da die Process Engine ebenfalls als ein Paket von Services im IS implementiert ist, kann der Prozessmanipulator auf einen großen Teil der Funktionalität der Prozessausführung zugreifen. Dazu werden API-Services verwendet, die eine Schnittstelle zu den Funktionen des IS bieten.

Im nächsten Kapitel werden die Flexibilisierungs-Varianten bestimmt, die vom Prozessmanipulator bereitgestellt werden sollen. Im übernächsten Kapitel werden dann Ansätze gesucht, die es ermöglichen die Flexibilisierungs-Varianten zu implementieren.

Kapitel 4

Prozessflexibilisierung

In diesem Kapitel wird der Begriff Prozessflexibilisierung für IT-gestützte Geschäftsprozesse vertieft und ein Rahmen ermittelt, um sie beschreiben und klassifizieren zu können. Im ersten Abschnitt 4.1 werden Taxonomien für die Prozessflexibilisierung vorgestellt. Anhand der dadurch gewonnenen Klassifikation und Terminologie werden in Abschnitt 4.2 konkrete Flexibilitätsmuster ausgearbeitet. Schließlich werden in Abschnitt 4.3 Anforderungen an ein Flexibilisierungs-System ermittelt.

4.1 Flexibilitäts-Taxonomien

In diesem Abschnitt werden zwei Taxonomien für Prozessflexibilität vorgestellt. Die durch die Taxonomien vorgestellten Klassen werden zudem anhand ihrer Kompatibilität mit ED-BPM bewertet. Mithilfe der Taxonomien sollen später Flexibilitäts-Muster gefunden werden, die für den Prozessmanipulator infrage kommen.

Die erste Taxonomie nach Regev betrachtet die Gegenstände und Eigenschaften der Prozess-Flexibilisierung. [WRR-2007b] Zum Beispiel sind das die Prozessmodel-Elemente als Gegenstände oder die Dauer der Flexibilisierung als eine Eigenschaft. Dagegen klassifiziert Schonenberg die Flexibilität anhand der praktischen Vorgehensweisen zur Prozessflexibilisierung. Etwa indem das Prozessmodell verändert oder der Sequenzfluss eines Prozesses manipuliert wird.

Zusätzlich zu den Beiden existieren noch weitere Prozessflexibilitäts-Taxonomien, die durch Schonenberg näher beleuchtet werden. [SMR$^+$-2007] Diese Taxonomien basieren jedoch zumeist auf Teilmengen der oben genannten Taxonomien und werden daher an dieser Stelle vernachlässigt.

4.1.1 Taxonomie nach Regev

Regev et al. schlägt vor, Prozessflexibilität in drei orthogonale Dimensionen aufzuteilen: Die Abstraktionsebene[1], das Subjekt der Veränderung[2] und die Eigenschaften der Veränderung[3]. [RSS-2006]

Regev gibt zwar die verschiedenen Änderungs-Merkmale an IT-gestützten Geschäftsprozessen in vollständiger Form an, beschreibt jedoch die vorgestellten Klassen nur oberflächlich. Der Ansatz definiert beispielsweise keine Verbindungen zwischen den vorgestellten Konzepten und eröffnet auch keine Verbindung zur praktischen Implementierung. [MAR-2008, Seite 102] Da die verschiedenen Klassen jedoch eine grundsätzliche Beschreibung des Umfangs von Änderungen an Geschäftsprozessen bieten, werden sie im Folgenden näher betrachtet und interpretiert.

Abstraktionsebene Die Abstraktionsebene spezifiziert die Ebene, auf welcher der Prozess verändert wird. Es wird zwischen Prozessmodell und Prozessinstanz unterschieden. Veränderungen können auf beiden Ebenen durchgeführt werden. Wird das Prozessmodell verändert, handelt es sich zumeist um langfristige Änderungen des gesamten Prozessablaufs zur Designzeit. Langfristige Änderungen können Strategiewechsel oder veränderte Prozess-Ziele sein. Werden einzelne Instanzen verändert, bedeutet dies, dass zur Ausführungszeit vom normalen Prozessablauf abgewichen werden muss, um diese spezifischen Prozessinstanzen zu einem (erfolgreichen) Ende zu führen.

Subjekt der Veränderung Diese Dimension unterscheidet zwischen den Subjekten des Prozesses, die verändert werden können. Dazu werden fünf Klassen definiert, die die verschiedenen Elemente eines Prozessmodells enthalten. Siehe Abbildung 4.1 auf Seite 28. Die funktionale Perspektive beschreibt die Funktion des Prozesses, genauer gesagt die Eingabe, der funktionale Ablauf mit nötigen Bearbeitungsschritten und das Ergebnis. Die Verhaltens-Perspektive enthält den Sequenzfluss eines Prozesses. Darunter fallen die Sequenzflüsse und die Art des Prozessablaufs. Die organisatorische Perspektive betrifft die am Prozess teilnehmenden Benutzerrollen, z. B. bestimmte Mitarbeiter für bestimmte Aktivitäten oder Zugriffsberechtigungen. Die operationale Perspektive beschreibt die technische Implementierung der Prozess-Aktivitäten, d. h. die Weise wie ein einzelner Arbeitsschritt ausgeführt wird. In der informationellen Perspektive werden die Daten-Objekte des Prozesses zusammengefasst.

[1] engl. Abstraction Level
[2] engl. Subject of Change
[3] engl. Property of Change

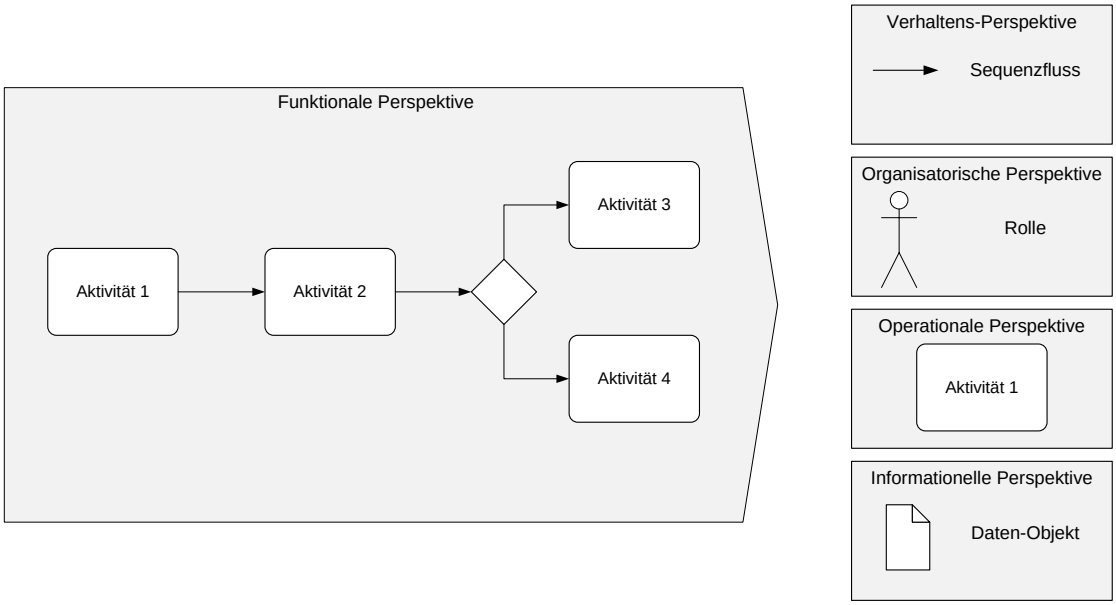

Abbildung 4.1: Subject of Change nach Regev [RSS-2006]

Eigenschaften der Veränderung In dieser Dimension werden die Eigenschaften der Änderungen zusammengefasst. Dazu zählt die Ausdehnung der Flexibilität, ob ein bestehendes Prozessmodell verändert wird oder ob ein neues Modell erstellt wird. Die Dauer einer Veränderung wird in temporär und permanent unterschieden. Ferner wird definiert ob alle bereits laufenden oder nur die zukünftigen Instanzen eines Prozesses unmittelbar oder verzögert verändert werden sollen. Letztlich wird unterschieden, ob eine Veränderung überraschend oder erwartet war. Die Eigenschaften dieser Klassen sind im Folgenden interpretiert:

Ausdehnung engl. *Extent*

inkremental – Es werden Elemente eines bestehendes Prozesses geändert. Die Änderungen basieren auf einem bestehenden Prozessmodell oder werden in diesem durchgeführt.

revolutionär – Es werden vollständig neue Elemente eines neuen Prozessmodells erstellt. Sämtliche Änderungen geschehen in diesem neuen Prozessmodell. Ein revolutionäres Prozessmodell kann auf einem alten Prozessmodell aufbauen oder ein vollkommen neues definieren.

Dauer engl. *Duration*

temporär – Die Änderung ist nur von eingeschränkter Dauer und gilt nur für einen zeitlichen eingeschränkten Zeitraum. Anschließend wird die Änderung ungültig oder zurückgenommen. In der Regel sind temporäre Änderungen nur Lösungen für temporäre Probleme.

permanent – Die Änderung ist mindestens bis zur nächsten Änderung gültig. Die Änderung bleibt bestehen, solange es keine weitere Änderung gibt, die diese ersetzt. Eine permanente Änderung kann die Lösung für ein generelles Problem darstellen.

Unmittelbarkeit engl. *Swiftness*[4]

unmittelbar – Änderungen werden unmittelbar durchgeführt, dadurch sind alle Instanzen von der Änderung betroffen. Die Änderungen werden auch auf die Instanzen angewandt, die bereits laufen.

verzögert – Die Änderungen werden verzögert durchgeführt, damit nur zukünftige Instanzen von der Änderung betroffen sind. Laufende Instanzen sind nicht betroffen, da die Änderung zurückgehalten wird. So können laufende Instanzen zu Ende gebracht werden und die Änderungen werden erst bei neuen Instanzen wirksam.

Vorwegnahme engl. *Anticipation*

unvorhergesehen – Die Änderungen sind ad hoc und waren nicht voraussehbar. Auf diese Weise konnten keine Vorkehrungen getroffen werden, die eine Änderung erleichtert hätten.

geplant – Die Änderungen wurden erwartet. Es wurden Anknüpfungspunkte für Veränderungen als Teil des Prozesses modelliert. Geplante Änderungen können Teil eines längeren und groß angelegten Prozess-Redesigns sein.

[4] dt. Geschwindigkeit oder Schnelligkeit. Aus dem Kontext heraus als Unmittelbarkeit übersetzt.

4.1.2 Taxonomie nach Schonenberg

Schonenberg et al. unterscheidet verschiedene Prozessflexibilität nach der Art ihrer Durchführung. Die Klassen „*Flexibilität durch Design*“[5], „*Flexibilität durch Abweichung im Sequenzfluss*“[6], „*Flexibilität durch Unterspezifizierung*“[7] und „*Flexibilität durch Veränderung*“[8] gehen jeweils von unterschiedlichen Einsatzzeitpunkten und Dimensionen aus. Während „Flexibilität durch Design“, vollständige Modelländerungen zur Design-Zeit klassifiziert, klassifiziert „Flexibilität durch Veränderung“ vollständige Modelländerungen, die laufende Prozessinstanzen beeinflussen. Bei „Flexibilität durch Abweichung im Sequenzfluss“ wird die Reihenfolge der Prozessausführung manipuliert. „Flexibilität durch Unterspezifizierung“ lässt bewusst Lücken im Prozessmodell, um sie zur Laufzeit auszufüllen. Schonenberg weist weiter darauf hin, dass die vorgestellten Taxonomien jedoch nur den Ablauf und Aufbau von Prozessen betrachten und nicht vollständig sind. [SMR$^+$-2008].

Um diese Taxonomien zu vervollständigen, werden später Muster der neu geschaffenen Klasse „Externalisierung“ betrachtet. Die Klasse enthält Muster zur Flexibilisierung mithilfe von dynamischen Daten-, Ressourcen- und Anwendungsreferenzierungen.

Flexibilität durch Design Diese Flexibilitäts-Klasse beschreibt den einfachsten und ursprünglichsten Ausdruck von Prozess-Flexibilität. Flexibilität durch Design ist dann möglich, wenn durch die Modellierungs-Notation parallelisiert werden kann und wenn Aktivitäten, Gateways und Sequenzflüsse modelliert werden können. Dabei wird angenommen, dass jeder Prozess durch sein Design bereits „flexibel“ sein kann. [SMR$^+$-2007] Jeder modellierte Pfad des Prozessmodells stellt dabei eine Art Flexibilität dar. Sobald ein Prozess je nach Ausgang einer Aktivität, unterschiedliche Sequenzflüsse aufrufen kann, die Alternativen zueinander sind, ist der Prozess per Design flexibel. Innerhalb dieser Klasse wird Flexibilität durch Gateways und bedingte Sequenzflüsse abgebildet, die je nach Prozessinstanz unterschiedliche Ausführungspfade durchlaufen. Jedoch ist Flexibilität, die zur Design-Zeit festgelegt wird, zu statisch um dynamisch flexibel zu sein. Muster dieser Klasse können nur auf bereits modellierte externe Ereignisse reagieren.

[5] engl. Flexibility by Design
[6] engl. Flexibility by Deviation
[7] engl. Flexibility by Underspecification
[8] engl. Flexibility by Change

Flexibilität durch Abweichung im Sequenzfluss Durch Abweichungen im Sequenzfluss wird kurzzeitig vom Ablauf einer Prozessinstanz abgewichen. Statt den Prozess wie durch das Prozessmodell definiert zu durchlaufen, werden Aktivitäten zu einem Zeitpunkt ausgeführt, der nicht dem vorhergesehenen Ablauf entspricht. Das bedeutet, dass Prozesselemente, die im üblichen Prozessablauf nicht an der Reihe wären, frei angesprungen werden können.

Flexibilisierungen dieser Art ermöglichen nur Änderungen an Prozessinstanzen und nicht an den Prozessmodellen. Die Abweichung im Sequenzfluss kann als eine Klasse angesehen werden, die frei bestimmen kann welche Aktivität als Nächstes ausgeführt wird.

Flexibilität durch Unterspezifzierung Diese Klasse geht davon aus, dass erst zur Laufzeit eines Prozesses klar ist, wie er aufgebaut sein soll. Gelöst wird dieses Problem, indem unklare Prozessmodell-Teile bewusst nicht spezifiziert werden und erst zur Laufzeit ausgefüllt werden. Der Prozess wird demnach nicht nur zur Laufzeit verändert, sondern auch erst zur Laufzeit komplettiert. Die äußerliche Struktur des Prozesses, wie Eingangs- und Ausgangspunkte, bleiben unverändert. Der Prozess bleibt bis auf wenige Stellen gleich und verändert nicht vollkommen seine Funktionsweise.

Die Idee hinter dieser Klasse geht davon aus, dass es an bestimmten Prozesselementen Flexibilitätspunkte gibt. Das sind Punkte, die beim Design des Modells zugewiesen und erst zur Laufzeit mit zusätzlichen Elementen fertig modelliert werden. Die Flexibilitätspunkte sind Platzhalter, die ersetzt werden müssen. „Spätes Binden“ und „Spätes Modellieren“ sind zwei Möglichkeiten, um Flexibilitätspunkte auszufüllen. Durch „Spätes Binden“ werden fertig definierte Prozessfragmente zur Laufzeit eingesetzt. Die Prozessfragmente sind zur Designzeit bereits bekannt und werden modelliert, bevor der Prozess ausgeführt wird. Bei der „Späten Modellierung“ wird das Prozessfragment erst zur Laufzeit modelliert und kann so völlig frei gestaltet werden.

Flexibilität durch Veränderung Diese Klasse umfasst Änderungen zur Laufzeit. In dieser Klasse können wie in der Design-Phase sämtliche Prozesselemente verändert, gelöscht oder hinzugefügt werden. Im Gegensatz zu den anderen Taxonomien kann hier das Prozessmodell vollständig verändert werden. Sie stellt also eine „Flexibilität durch Design“-Variante zur Laufzeit dar.

Betroffenen Instanzen müssen anschließend abgebrochen, neu gestartet oder in das neue Modell überführt werden. Es wird unterschieden, ob es sich um eine temporären Veränderung [9] oder um eine fortdauernde Veränderung [10] handelt. Weiterhin wird betrachtet,

[9] engl. Momentary Change
[10] engl. Evolutionary Change

zu welchem Prozessausführungs-Zeitpunkt eine Veränderung durchgeführt werden kann und wie eine laufende Prozessinstanz darauf reagiert. Es wird zwischen den beiden Zeitpunkten „Entry Time“ und „On the fly“ unterschieden. Wobei Ersterer Änderungen nur zum Beginn eines Prozesses erlaubt und Letzterer zu jedem Zeitpunkt des Prozesses. Zudem wird differenziert welche „Migration Strategy“ vorliegt, d. h. wie mit laufenden und betroffenen Instanzen verfahren wird. Mögliche Varianten sind dabei das Abbrechen, Neustarten, Weiterlaufen lassen und das gezielte Wiedereinsetzen des Prozesses.

4.1.3 Beziehung zwischen den Taxonomien und ED-BPM

Dieser Abschnitt beschreibt die Beziehung zwischen den vorgestellten Taxonomien und ED-BPM. Im Kontext dieser Arbeit und im ADiWa-Projekt wird Geschäftsprozessflexibilität als unmittelbare und nicht dauerhafte Prozessadaption verstanden. Bei der Diskussion darüber, was ED-BPM ist und wie ED-BPM im Kontext der Aufgabenstellung aufgefasst wird, lassen sich die folgenden drei Rahmenbedingungen aufstellen:

1. *Prozesse sollen auf Instanzebene flexibilisiert werden.*
   Die nötigen Flexibilisierungen am Prozess sollen nur in Ausnahmefällen gelten, die von Ereignissen abhängig sind.

2. *Das Prozessmodell soll für nachfolgende Instanzen unverändert bleiben.*
   Die Änderungen sind immer speziell und tief mit der betroffenen Instanz verbunden. Sie lassen sich nicht auf die Menge aller Prozesse anwenden, wenngleich sie durchaus mehrere Instanzen betreffen können.

3. *Prozessinstanzen sollen jederzeit, vollständig und ohne vorherigen Eingriff flexibilisiert werden können.*
   Wie, wann und auf welche Weise ein Problem gelöst wird, ist nicht definiert. Daher ist es nicht möglich, vorbereitetende Maßnahmen für eine Flexibilisierung zu treffen.

Die drei Rahmenbedingungen schließen evolutionäre Änderungen am Prozessmodell durch Prozessverbesserung oder Prozessinnovation aus (vgl. Abschnitt 2.2 auf Seite 10). Änderungen dieser Art sind grundsätzliche Flexibilisierungen am Prozessmodell, die nicht auf einzelne Ereignisse reagieren. Solche Flexibilisierungen sollen nicht ereignis-gesteuert durchgeführt werden.

Beziehung zwischen der Regev-Taxonomie und ED-BPM

Das folgende dreidimensionale Karnaugh-Veitch-Diagramm (KV-Diagramm) in Abbildung 4.2 auf Seite 34 stellt mithilfe der Regev-Taxonomie die Eigenschaften von ED-BPM dar. Das KV-Diagramm bietet die Möglichkeit, mehrere Auswahlmöglichkeiten direkt ausschließen zu können. Der Vorteil besteht darin, nicht jedes Feld des Diagramms einzeln beurteilen zu müssen, sondern von den Spalten und Zeilen auf die betreffenden Felder schließen zu können. Die Flexibilisierungen die den Kriterien Abstraktionsebene, Subjekt und Eigenschaften entsprechen (mit $\times$ markiert) sollen später gefunden werden.

Zuerst soll das KV-Diagramm nach der *Abstraktionsebene* der Flexibilisierung aufgelöst werden. Ein ereignis-gesteurtes System ist dynamisch. Diese Dynamik bezieht sich auf die von Modellen abgeleiteten Instanzen innerhalb eines Systems. Flexibilisierungen zur Designzeit, also auf Modell-Ebene, sind Teil aller Geschäftsprozessausführungsumgebungen und damit kein Alleinstellungsmerkmal von ED-BPM und dieser Arbeit. Aus diesem Grund werden alle Flexibilisierungen, die auf Modell-Ebene ablaufen vernachlässigt. Ziel dieser Arbeit sind Flexibilisierungen zur Ausführungszeit.

Im nächsten Schritt wird das Diagramm nach dem *Subjekt* der Veränderung aufgelöst. Da aber alle Subjekte Teil eines Prozesses sind und daher auch beleuchtet werden sollen, wird nichts ausgeschlossen. Alle Subjekte werden im folgenden Schritt miteinbezogen.

Im letzten Schritt werden die *Eigenschaften* der Prozessflexibilisierung untersucht. Für jede der Eigenschaften, Ausdehnung, Dauer, Unmittelbarkeit und Vorwegnahme, soll eine Ausprägung ausgewählt werden. Revolutionäre Änderungen werden nicht durch Ereignisse gesteuert werden, da sie meist große Änderungen mit sich bringen. Die flexiblen Änderungen bauen auf einem bestehenden Prozessmodell auf und sind daher inkrementtal. Da die Änderungen immer für bestimmte Ereignisse eine kurzfristige und nicht dauerhafte Lösung erwirken sollen, sind sie temporär und nicht von permanenter Dauer. Weiterhin lauten die Anforderungen nach „Ad-hoc“-Änderungen an den Instanzen. Demnach werden keine verzögerten Flexibilisierungen gefordert, sondern Unmittelbare. Zudem sind zudem unvorhergesehene Änderungen an Prozessinstanzen gefordert. Daher können geplante Änderungen vernachlässigt werden.

Mithilfe des KV-Diagramms konnten die Modell-Abstraktionsebene und die Flexibilitäts-Ausprägungen revolutionär, permanent, verzögert und geplant ausgeschlossen werden. Obwohl sie für diese Arbeit nicht relevant sind, könnten sie für spätere Entwicklungen durchaus von Bedeutung sein. Bis dahin unterstützt eine vollständige ED-BPM-Implementierung idealerweise die im Diagramm identifizierten Flexibilisierungen.

				Eigenschaften							
				Ausdehnung		Dauer		Unmittelbarkeit		Vorwegnahme	
				inkremental	revolutionär	temporär	permanent	unmittelbar	verzögert	unvorhergesehen	geplant
Abstraktionsebene	**Modell**	*Subjekt*	Funktional								
			Organisatorisch								
			Verhalten								
			Informationell								
			Operational								
	Instanz	*Subjekt*	Funktional	×		×		×		×	
			Organisatorisch	×		×		×		×	
			Verhalten	×		×		×		×	
			Informationell	×		×		×		×	
			Operational	×		×		×		×	

Abbildung 4.2: Beziehung zwischen der Regev-Taxonomie und ED-BPM

Beziehung zwischen der Schonenberg-Taxonomie und ED-BPM

Die Anforderungen an Flexibilisierungen wurden bereits im Vergleich von Regev und ED-BPM zusammengefasst. Mithilfe der Abbildung 4.3 wird eine Vorauswahl getroffen, mit welchen technischen Ansätzen Flexibilisierung erreicht werden kann. Wie in der Abbildung aufgezeigt, bieten nur die bereits bekannten Schonenberg-Klassen „Veränderung“ und „Abweichung“ vollständige Flexibilisierung zur Laufzeit im Verständnis dieser Arbeit.

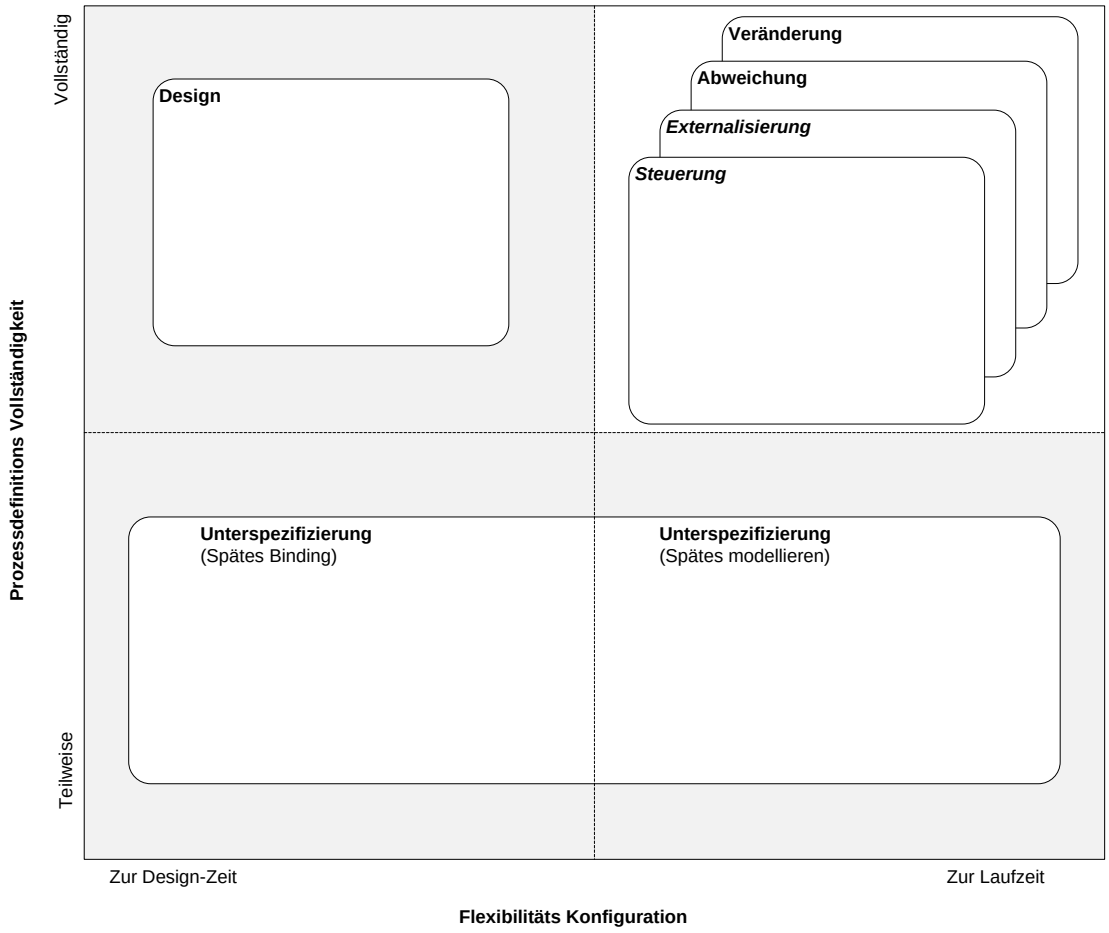

Abbildung 4.3: Flexibilitäts-Typen Spektrum

Aufbauend auf [SMR$^+$-2007]

Ungeeignete Klassen sind grau hinterlegt und können bereits an dieser Stelle vernachlässigt werden. Obwohl auch die Klasse „Unterspezifizierung“ zur Laufzeit angewandt werden kann, ist diese Klasse zu statisch. Bei der Unterspezifizierung kann die Prozessinstanz nur an vorher festgelegten Punkten verändert werden. Sie ähnelt daher dem

vorgestellten Döhring-Ansatz (vgl. Abschnitt 2.2.1 auf Seite 12). Die Flexibilität zur Design-Zeit („Design“) ist zwar vollständig flexibel, bei der dynamischen Flexibilisierung von Instanzen aber nicht zu gebrauchen. Wie der Name bereits sagt, wird die Flexibilität nur durch Änderungen am Modell zur Designzeit durchgeführt.

Zusätzlich zu den bekannten Schonenberg-Klassen, wurden die Klassen „Externalisierung“ und „Steuerung“ aufgenommen. Beide Klassen werden im folgenden Abschnitt erstmals vorgestellt. Weiterhin werden die Klassen „Veränderung“ und „Abweichung“ weiter betrachtet.

4.2 Flexibilitätsklassifkation

Der folgende Abschnitt geht auf die Flexibilitäts-Klassen Abweichung und Veränderung ein. Diese Klassen wurden im vorhergehenden Abschnitt als ED-BPM-geeignet identifiziert und werden an dieser Stelle noch weiter präzisiert. Zusätzlich werden die neuen Klassen „*Externalisierung*“ und „*Steuerung*“ spezifiziert. Zu allen als geeignet identifizierten Klassen werden konkrete Flexibilitäts-Muster vorgestellt. Zur Klassifizierung der Muster werden diese in die vorgestellte Taxonomie gegliedert und nach einer näheren Beschreibung der Klasse aufgeführt. Flexibilitäts-Muster sind wiederholende Muster, mit denen Flexibilität erreicht werden kann. Die folgenden Muster ergeben sich aus den von Weber et al., Mulyar et al. und Schonenberg et al. vorgestellten Varianten.

Weber beschreibt detaillierte Muster, die mögliche Änderungs-Varianten auf Modellebene sind. Zum Beispiel sind das Hinzufügen, Ersetzen und Löschen von Aktivitäten. [WRR-2007b] Andere Aspekte der Prozessmodellierung, wie das Daten-Objekt oder Externalisierung durch Services werden nicht betrachtet. Schonenberg beschreibt für die selbst-definierte Klasse „Flexibilität durch Veränderung“ Rahmenbedingungen und Einschränkungen, geht aber nicht auf konkrete Muster ein. Bei der Klasse „Flexibilität durch Abweichung im Sequenzfluss“ werden die möglichen Änderungen hingegen detailliert beschrieben. [SMR$^+$-2007] Zudem greift Mulyar et al. die von Schonenberg definierten Flexibilitäts-Klassen auf und definiert für jede der vier Klassen zusätzliche „Flexibility Patterns“. [MAR-2008] Soweit sinnvoll, werden eigene Muster entwickelt. Insbesondere bei den Klassen „Flexibilität durch Externalisierung“ und „Flexibilität durch Prozess-Steuerung“, die von keinen der Quellen diskutiert werden.

Zur Beschreibung der Muster wurden Begriffe aus der Regev-Taxonomie verwendet. Nachdem das Muster grundsätzlich beschrieben wurde, folgt ein Beispiel. Sofern kein reales Beispiel vorliegt, sind diese Beispiele konstruiert. Zudem handelt es sich dabei um Beispiel-Fälle, die aufgrund ihrer geringen Auftritts-Wahrscheinlichkeit nicht im Prozessmodell modelliert werden würden.

4.2.1 Flexibilität durch Prozess-Steuerung

Die Prozess-Steuerung ändert den Zustand einer Prozessinstanz, auf diese Weise kann auch Prozessflexibilität bereitgestellt werden. Wird ein Prozess gestartet, pausiert oder wiederaufgenommen kann dies bereits eine flexible Lösung für ein Problem sein. Die folgenden Muster beschreiben Szenarien, in denen alleine mit den Steuerungs-Funktionen der Prozessausführung ein Prozess flexibilisiert werden kann.

Starten (Verhalten)
Eine neue Instanz eines Prozesses wird erstellt.
Beispiel: Weil ein Transportflugzeug zur Landung ansetzt, soll ein neuer Import-Warenlager-Prozess gestartet werden.

Abbrechen (Verhalten)
Eine fehlerhafte oder überflüssige Prozessinstanz wird beendet.
Beispiel: Aufgrund einer fehlerhaften markierten Sendung wird irrtümlich ein entsprechender Prozess gestartet. Der Fehler wird erkannt und der Prozess abgebrochen.

Pausieren (Verhalten)
Der Prozess wird an der aktuellen Stelle angehalten.
Beispiel: Bei der Abwicklung durch den Zoll kommt es wegen einer unbekannten und dadurch potenziell gefährlichen Sendung zu erheblichen Verspätungen. Die Transportunternehmen werden gebeten, ihre Prozesse vor oder während des Transports anzuhalten, um einen Rückstau vor der Warenkontrolle zu verhindern.

Wiederaufnehmen (Verhalten)
Ein zuvor pausierter Prozess wird wiederaufgenommen.
Beispiel: Die unbekannte Sendung hat sich als unbedenklich erwiesen. Die pausierten Prozesse können wiederaufgenommen werden.

4.2.2 Flexibilität durch Abweichung im Sequenzfluss

Diese Flexibilitäts-Klasse weicht vom vorgegebenen Ausführungspfad ab, ohne das Prozessmodell zu verändern. Der Sequenzfluss wird immer auf Instanz-Ebene verändert. Durch die Abweichung kann jedoch nur der Prozessablauf und nicht der funktionale Prozessaufbau verändert werden. Die folgenden Muster basieren auf den Flexibility-Pattern *Undo*, *Redo*, *Skip*, *Additional Instance* und *Invoke* von Schonenberg et al. [SMR$^+$-2007]

Wenn im Folgenden von Prozessteilen gesprochen wird, stehen diese für ein oder mehrere Prozesselemente mit assoziierten Sequenzflüssen.

Rückgängig [11] (Verhalten) Ein bereits ausgeführter Prozessteil wird rückgängig gemacht, bzw. der Aufruf einer Aktivität zurückgenommen.
Beispiel: Eine komplette Palette wurde vom Versender irrtümlich mit leeren Containern bestückt. Weil dieser Container bereits beim Zoll gemeldet, und als zollpflichtig eingestuft wurden, soll die Meldung rückgängig gemacht werden.

Wiederholen [12] (Verhalten) Ein vorangegangener Prozessteil soll erneut ausgeführt werden, bzw. eine bereits aufgerufene Aktivität wiederholt werden.
Beispiel: Durch die fehlerhafte Codierung von kambodschanischen Schriftzeichen im Namen eines Auftraggebers wurden die Ladelisten nur zerstückelt übergeben. Für alle Lieferungen dieses Versenders soll die Übergabe erneut durchgeführt werden.

Überspringen [13] (Verhalten) Nicht essenzielle Aktivitäten werden übersprungen, um ein Endergebnis schneller zu erwirken oder um Aktivitäten gezielt nicht auszuführen.
Beispiel: Durch die verspätete Ankunft eines Flugzeuges soll die Zwischenlagerung einer Sendung übersprungen und direkt mit dem Transport zum Kunden begonnen werden. Dadurch soll der Kunde die Sendung im versprochenen Zeitrahmen erhalten.

Zusätzliche Instanz [14] (Verhalten) Ein Prozessteil bzw. eine Aktivität wird mehrfach gestartet.
Beispiel: Für den Transport einer Sendung zum Kunden reicht der beauftragte Transport nicht aus. Zusätzlich zum bereitstehenden Transporter soll ein zweiter der gleichen Art beauftragt werden.

Sprung [15] (Verhalten) Ein Prozessteil, der nicht an der Reihe ist, soll ausgeführt werden, bzw. eine Aktivität soll vor ihrem eigentlichen Aufruf ausgeführt werden.
Beispiel: Weil bereits vor der Anmeldung beim Zoll eine starke zeitliche Verzögerung abzusehen ist, soll der Transport ins Zwischenlager schon vor dem Transport zur Warenkontrolle beauftragt werden.

[11] Undo
[12] Redo
[13] Skip
[14] Additional Instance
[15] Invoke Task

4.2.3 Flexibilität durch Externalisierung

Bei der Flexibilität durch Externalisierung werden die zu flexibilisierenden Elemente eines Prozesses modularisiert und außerhalb des Prozesses implementiert. Bei den Mustern dieser Klasse werden Ausführung und Funktionalität entkoppelt. Die Ausführung liegt nach wie vor beim Prozess. Der Sequenzfluss läuft durch den Prozess und aktiviert die relevanten Prozess-Elemente.

Diese Klasse trägt den Namen Externalisierung, weil die Funktionalität außerhalb des Prozesses liegt. So kann indirekt das Verhalten eines Prozesses geändert werden, indem die externe Funktionalität ausgetauscht oder verändert wird. In dieser Klasse wurden zwei Muster identifiziert.

Dynamische Service-Referenz (Operational) Die Referenz auf einen Service wird zur Laufzeit verändert. Eine Aktivität soll durch eine andere Implementierung gelöst werden, als vorgesehen.
Beispiel: Eine verspätete Sendung soll nicht erst in das Zwischenlager transportiert werden, sondern direkt zum Kunden. Die Aktivität „Transport-Auftrag zum Zwischenlager“ verweist auf einen Service der Transporte innerhalb des Flughafengeländes beauftragt. Weil die Sendung direkt zum Kunden transportiert werden soll, wird stattdessen die Referenz auf den Service zur Beauftragung von externen Transporten gelegt.

Flexibler Service (Operational) Die Service-Implementierung entscheidet aufgrund der Instanz, welches Verhalten bereitgestellt wird. So kann für eine bestimmte Prozessinstanz ein verändertes Verhalten gewählt werden.
Beispiel: Aufgrund einer System-Überlastung beim Zoll sollen für einen kurzen Zeitraum nicht mehr alle Ladelisten über den Ladelisten-Service des Zolls gemeldet werden. Stattdessen sollen für die Dauer der Wartungsarbeiten, Ladelisten für Sendungen aus Asien als E-Mail gesandt werden. Der Service, der die Übergabe der Ladeliste implementiert, wird so umgeschrieben, dass er für entsprechenden Instanzen E-Mails versendet, statt den entfernten Zoll-Service aufzurufen.

4.2.4 Flexibilität durch Veränderung

Zu dieser Klasse zählen alle Muster, die das Prozessmodell vollständig und uneingeschränkt verändern können. Also in demselben Maße, wie es das Werkzeug zur Prozessmodellierung selbst könnte.

Die mit Fußnoten versehenen Muster basieren auf den „Adaption Patterns“ die von Weber und den „Flexibility Patterns“ die von Mulyar et. al definiert wurden. [WRR-2007a]

[MAR-2008, Seite 98] Da diese Muster nur das Verhalten und den funktionalen Aufbau verändern, kommt das Muster „Daten-Objekt verändern“ hinzu.

Hinzufügen [16] (Funktional) Eine nicht modellierte Aktivität soll durchgeführt werden. Ein Prozessteil wird in das Modell eingefügt. Dabei kann es nach einem Element oder parallel zu einem Element eingefügt werden. Bei Bedarf werden die dadurch notwendigerweise ebenso hinzugefügten Sequenzflüsse noch um Bedingungen erweitert.

Beispiel: Sendungen einer bestimmten Fluglinie wurden in den letzten Tagen vermehrt beschädigt. Nach der Zoll-Abwicklung sollen die entsprechenden Sendungen intensiv auf äußerliche Schäden untersucht werden.

Löschen [17] (Funktional) Eine Aktivität soll übersprungen oder gelöscht werden. Dazu soll ein Prozessteil aus dem Modell gelöscht werden. Alle zum und vom Element führenden Sequenzflüsse werden gelöscht.

Beispiel: Da Eile geboten ist, soll auf die aufwendige Zwischenlagerung der Sendung verzichtet werden. Die Aktivitäten zur Zwischenlagerung „Transport-Auftrag zum Zwischenlager“ und „Einlagerung im Zwischenlager“ werden gelöscht. Die verbliebenen Sequenzflüsse werden zur Aktivität „Transport-Auftrag zum Kunden“ weitergeleitet.

Bewegen [18] (Funktional) Ein Prozessteil wird von einer Stelle des Modells an eine andere bewegt und wird so zu einem anderen Zeitpunkt ausgeführt.

Beispiel: Ladelisten mit Sendungen aus einem mit einem Handelsembargo belasteten Land sollen noch vor der Zoll-Anmeldung übergeben werden.

Ersetzen [19] (Funktional) Ein Prozessteil wird durch einen Anderen ersetzt. Hierzu kann es kommen, wenn eine Aktivität nicht mehr angemessen ist und ersetzt werden soll oder wenn eine Aktivität entscheidend verändert wird.

Beispiel: Der versendende Spediteur bemerkt noch vor der Landung des Flugzeuges, dass eine fehlerhafte Sendung aufgegeben wurde. Unmittelbar nach der Landung und noch vor dem Empfang, soll die Sendung wieder in das Flugzeug geladen werden.

[16] AP1: Insert Process Fragment (Weber et al.) und FP26: Task Insertion (Mulyar et al.)
[17] AP2: Delete Process Fragment und FP22: Task Elimination
[18] AP3: Move Process Fragment und FP18: Reordering
[19] AP4: Replace Process Fragment

Vertauschen [20] (Funktional) Die aufeinanderfolgende Anordnung zweier Aktivitäten soll vertauscht werden, bzw. zwei Prozessteile sollen miteinander vertauscht werden.

Beispiel: Zur Besseren Kontrolle des Staus vor der Warenkontrolle bittet der Zoll, die Warenliste vor dem Transport zur Warenkontrolle zu übergeben. Dazu werden die Aktivitäten „Transport zur Warenkontrolle“ und „Übergabe der Ladeliste“ miteinander vertauscht.

Schleife einfügen [21] (Funktional) Ein Prozessteil soll in einer Schleife wiederholt werden.

Beispiel: Die Anmeldung beim Zoll ist durch den großen Betrieb zusammengebrochen und funktioniert nicht mehr vorhersagbar. Da eine Sendung hohe Priorität hat, soll die Anmeldung immer wieder versucht werden.

Parallelisierung [22] (Funktional) Prozessteile, die zuvor sequenziell ablaufen sollten, werden parallel geschaltet.

Beispiel: Der Transport-Auftrag soll zur Zeitersparnis zeitlich mit der Einlagerung im Zwischenlager parallel geschehen.

Ein- und Austritt ändern [23] (Funktional) Das Start- oder Ende-Ereignis eines Prozesses wird verändert.

Beispiel: Der Kunde ist mangels Zahlungsfähigkeit nicht mehr berechtigt, die Sendung zu empfangen. Deswegen endet der Prozess mit der verlängerten Zwischenlagerung der Sendung.

Bedingte Ausführung [24] (Funktional) Ein Prozessteil soll nur ausgeführt werden, wenn eine neuerliche Bedingung zutrifft. Dieses Muster unterscheidet sich zum Muster „Sequenzflussbedingung ändern“ derart, dass mit der bedingten Ausführung große Prozessteile von der Ausführung abgehalten werden.

Beispiel: Der gesamte Warenlager-Prozess soll nach der Ankunft des Flugzeuges nur dann ausgeführt werden, wenn das sendende Speditionsunternehmen zahlungsfähig ist.

Sequenzfluss ändern [25] (Funktional) Prozessteile, die zuvor nicht verbunden waren, sollen durch zusätzliche Sequenzflüsse erweitert werden bzw. bestehende Sequenzflüsse gelöscht werden.

Beispiel: Obwohl eine Sendung nicht zollpflichtig ist, soll die Ladeliste zur Kontrolle

[20] AP5: Swap Process Fragment und FP18: Reordering
[21] AP8: Embed Process Fragment in Loop
[22] AP9: Parallelize Activities
[23] FP1: Alternative Entry Points
[24] AP10: Embed Process Fragment in Conditional Branch
[25] AP11: Add Control Dependency / AP12: Remove Control Dependency und FP14: Choice Insertion

übergeben werden. Nach dem Transport zum Zwischenlager wird ein zusätzlicher Sequenzfluss zur Aktivität „Übergabe der Warenliste“ eingefügt.

Sequenzflussbedingung ändern [26] (Funktional) Für einen Ablauf sollen neuerliche Bedingungen eingeführt werden. Bestimmte Prozessteile sollen nur ausgeführt werden, wenn Bedingungen erfüllt sind.
Beispiel: Sendungen aus dem EU-Überseegebiet Französisch-Guyana sollen, obwohl regulär nicht zollpflichtig, zur Warenkontrolle gebracht werden. Dazu wird die Bedingung „Nicht zollpflichtig“ folgendermaßen erweitert: „Nicht zollpflichtig und nicht aus Französisch Guyana“. Die Bedingung „Zollpflichtig“ wird zu „Zollpflichtig oder aus Französisch Guyana“.

Daten-Objekt verändern (Informationell) Eine zum Prozessbeginn definierte Variable muss verändert werden. Das Daten-Objekt soll bearbeitet werden.
Beispiel: Der Kunde wünscht die Sendung nicht an ihr eigentliches Ziel, sondern an ein anderes geliefert zu bekommen. Dazu wird die Kunden-Adresse im Daten-Objekt angepasst.

4.3 Zusätzliche Anforderungen an die flexible Prozessausführung

Die flexible Geschäftsprozessausführung besteht nicht nur aus der Entwicklung geeigneter Methoden, sondern auch aus Anforderungen, die zum Einsatz im produktiven geschäftlichen Umfeld unbedingt notwendig sind. Ein Prozessflexibilisierungs-System muss z. B. nicht nur in der Lage sein, Prozesse nach Mustern zu verändern, sondern auch Prozesszustände und das ausführende System konsistent zu halten. Diese Anforderungen können in elementare und erweiternde Features gegliedert werden.

Weber et al. definiert drei elementare Features für die flexible Prozessausführung. Die identifizierten Features sind die Modell-Versionierung und Instanz-Migration, die Unterstützung von Ad-hoc-Veränderungen und die Zusicherung, dass ein Prozess nach der Flexibilisierung noch konsistent ist. [WRR-2007a]

Modell-Versionierung und Instanz-Migration Die Versionierung und Instanz-Migration beschreibt den Umgang mit Instanzen, deren Prozessmodell flexibilisiert wird. Wenn ein Prozessmodell verändert wird, hat dies Auswirkungen auf die abgeleiteten Instanzen. Unterschieden wird dabei zwischen laufenden und zukünftigen Instanzen. Zukünftige Instanzen, also Instanzen, die nach der Flexibilisierung gestartet werden, sind dabei bei

[26] AP13: Update Condition und FP14: Choice Insertion

Veränderungen eher unkritisch. Kritischer sind laufende Instanzen, da bei ihnen unterschieden werden muss, ob die Flexibilisierung angewandt werden soll oder nicht. Dabei ergeben sich zwei mögliche Szenarien, je nachdem, ob das Prozessausführungssystem eine Versionierung von Prozessen unterstützt. Versionierung bedeutet hier, dass es möglich ist, mindestens zwei Versionen eines Prozessmodells parallel im System zu hinterlegen und für die Prozessausführung zugänglich zu machen.

Ohne Versionierung ist es dem Prozessmodellierer überlassen, externe Versionen des Prozesses zu speichern. Im Prozessausführungssystem selbst kann zu jedem Zeitpunkt immer nur eine Version abgelegt und aktiviert sein. Das heißt für laufende wie zukünftige Prozessinstanzen, dass sie bei einer Flexibilisierung in das flexibilisierte Prozessmodell migriert werden müssen. Durch die Migration laufender Instanzen können Inkonsistenzen entstehen, die im schlimmsten Fall dazu führen, dass der Prozess aufgrund eines Fehlers abgebrochen wird.

Mit Versionierung werden flexibilisierte Prozessmodelle in einer anderen Version abgelegt, als die zuvorgehende. Das Problem von Instanzen, die auf einer alten Version basieren, lässt sich auf verschiedenen Wegen lösen. [SMR$^+$-2007] Im einfachsten Fall werden sämtliche betroffenen Instanzen abgebrochen. Die zweite Variante unterbricht die betroffenen Instanzen und überführt sie in die neue Version. Dadurch dass dank der Versionierung mehrere Versionen des Modells bestehen, können Prozessinstanzen auch unverändert ablaufen und so ihre Konsistenz wahren. Alternativ dazu kann die Versionierung eine Funktion mitbringen, die es erlaubt, laufende Prozessinstanzen in eine neuere Version zu migrieren. Dazu werden alle Prozessinstanzen, die zur neuen Version kompatibel sind, von der alten Version zur Laufzeit in die neue Version überführt. Kompatibel ist eine Prozessinstanz dann, wenn die Ausführungs-Historie, das heißt die Schritte die durchlaufen wurden und die Daten die generiert wurden, sich auch mit der neuen Version reproduzieren ließen. Ohne Migration der Instanz kann es zu Problemen kommen. Zum Beispiel wird eine Prozessinstanz Aktivität A1 ausgeführt, die von der Aktivität A2 nachgefolgt wird. A2 wird nun für alle laufenden Prozessinstanzen gelöscht. Weil die Prozessinstanz nicht migriert wurde, versucht sie als Nächstes die Aktivität A2 aufzurufen, die im System nicht mehr hinterlegt ist. Die Prozessinstanz bricht mit einem Laufzeitfehler ab.

Ad-hoc-Zugriff und Änderungen auf Instanz-Ebene [27]Ein ED-BPM-konformes Prozessausführungssystem muss in der Lage sein, Ad-hoc-Veränderungen durchzuführen. Ad-hoc heißt in diesem Zusammenhang, dass Flexibilisierungen unmittelbar, unvorhergesehen und zu jedem Zeitpunkt auf Instanzebene geschehen können. Dazu zählen das

[27]Support for Ad-hoc Changes / Support for Instance-specific Process Changes

Hinzufügen, Löschen und Bearbeiten von Prozessteilen. Beispielsweise können einer laufenden Prozessinstanz beliebig viele Aktivitäten entnommen oder hinzugefügt werden. Dabei sollte der Prozessmodellierer nicht durch Zeit und Ort der Flexibilisierung eingeschränkt sein.

Dieses Feature ist Deckungsgleich mit der Problemstellung dieser Arbeit und wurde daher bereits in den Kapiteln zuvor spezifiziert. Zudem sollte nach Weber die Prozessausführung in der Lage sein, Teile des Prozesses unterspezifiziert zu lassen und sie zur Laufzeit ausführen. Die Unterspezifizierung von Prozessteilen ist nicht Teil dieser Arbeit.

Konsistenzprüfung [28] Dieses Feature sorgt dafür, dass auch nach einer Flexibilisierung ein syntaktisch korrektes und konsistentes Prozessmodell zurückbleibt. Ein Prozessmodell ist dann syntaktisch korrekt, wenn es nach der Flexibilisierung ausführbar bleibt und dieselben Anforderungen erfüllt, die es unflexibilisiert erfüllt hat. Zum Beispiel müssen auch in einem flexibilisierten Modell alle Sequenzflüsse syntaktisch korrekt einen Anfang und ein Ende haben.

Ein konsistentes Prozessmodell bedeutet, dass es auch nach der Flexibilisierung ohne Fehler auszuführen ist. Zu einem Laufzeitfehler kann es dann kommen, wenn z. B. eine einzelne Aktivität aus einer Reihe von voneinander abhängigen Aktivitäten unzulässig flexibilisiert wird. Eine solche Reihe von stark voneinander abhängigen Aktivitäten wird als Transaktion bezeichnet. Ein Fehler entsteht, wenn die Eingangsdaten einer Aktivität durch eine vorangegangene Aktivität nicht mehr bereitgestellt werden. Das kann dann passieren, wenn die vorhergehende Aktivität gelöscht oder so geändert wurde, dass die gewünschten Daten nicht mehr generiert werden. Wird z. B. am Prozess A eine Flexibilisierung an den Aktivitäten 2 und 3 durchgeführt, hat das für eine Prozessinstanz P1, die an Aktivität 1 steht, zumindest für die Konsistenz ihrer Ausführung keine Auswirkungen. Für Prozessinstanz P2, die Aktivität 2 abgeschlossen hat und mit Aktivität 3 beginnen will, hingegen schon. Die beiden Aktivitäten können sich in der Zwischenzeit so verändert haben, dass sie unter völlig veränderten Annahmen laufen. Beispielsweise liefert die durchlaufene Aktivität 2 nicht mehr die Daten, die die flexibilisierte Aktivität 3 in ihrem neuen Zustand von ihr erwartet. Eine Prozessausführung muss daher dafür sorgen, dass Transaktionen nur als Ganzes oder gar nicht verändert werden.

Flexibilisierungen, die syntaktische Bedingungen und Transaktionen verletzten, sollten stets zurückgewiesen und nicht durchgeführt werden.

[28] Correctness of Change

4.4 Zusammenfassung

In diesem Kapitel wurden die beiden Flexibilitäts-Taxonomien nach Regev und Schonenberg vorgestellt. Anhand der Schonenberg-Taxonomie konnten Klassen ermittelt werden, nach denen Flexibilität anhand ihrer Ausführung gegliedert werden kann. Neben den Klassen „Veränderung“ und „Abweichung im Sequenzfluss“ wurden die beiden Klassen „Abweichung im Sequenzfluss“ und „Externalisierung“ eingeführt. Die Klassen „Veränderung durch Design“ und „Unterspezifizierung“ wurden ausgelassen, da sie nicht den ED-BPM-Rahmenbedingungen entsprechen.

Innerhalb der ausgewählten Klassen wurden Muster gefunden, die mithilfe der Regev-Taxonomie in die Informationelle, Funktionale und Operationale Ebene sowie die Verhaltens-Ebene gegliedert wurden. Die organisatorische Ebene wird im Folgenden ignoriert, da für sie kein Muster identifiziert wurde.

Neben den Mustern, die die Bandbreite an möglichen Flexibilisierungen festlegen, wurden elementare Features für das Prozessflexibilisierungs-System aufgeführt. Mit der Versionierung werden mehrere Ausprägungen eines Prozessmodelle zur gleichen Zeit unterstützt, die einander nicht beeinflussen. Durch Ad-hoc-Änderungen kann eine Prozessinstanz jederzeit und überall flexibilisiert werden. Die Konsistenzprüfung stellt sicher, dass ein Prozessmodell auch nach der Flexibilisierung in einem korrekten und ausführbaren Zustand bleibt. Der Prototyp soll später nach diesen Kriterien bewertet werden.

Im nächsten Kapitel werden Lösungsansätze für webMethods gesucht, mit denen die Muster implementiert werden können.

Kapitel 5

Lösungsansätze für webMethods

In diesem Kapitel werden konkrete Lösungsansätze vorgestellt, die im Kontext von webMethods bereits bestehen oder implementiert werden können. Es handelt sich dabei zumeist um Ansätze, die auf bereits existierenden webMethods-Schnittstellen aufbauen oder diese im Hinblick auf Flexibilität erweitern.

Die vorgestellten Ansätze sind funktional abgetrennte Teilaspekte des Gesamt-Komplexes „Flexibilisierung nach ED-BPM“. Jeder Ansatz findet jeweils Anwendung in einem Feld der Flexibilisierung. Zum Teil sind diese Abtrennungen unfreiwillig und technisch bedingt. Im Prozessmanipulator-Prototyp werden einige der Teilaspekte zu einer Gesamt-Implementierung zusammengefügt, um möglichst viele Aspekte der Prozessflexibilisierung abzudecken.

5.1 Prozess-Steuerung

Der Ansatz „Prozess-Steuerung“ deckt alle Zustände des Prozessablaufs ab und ist Teil der gleichnamigen Klasse „Prozess-Steuerung“. Die Prozessausführung von webMethods kennt zahlreiche Zustände für eine Prozessinstanz. Neben internen Zuständen sind die Zustände „läuft“ bzw. „wiederaufgenommen“, „pausiert“, „abgebrochen“ und „abgeschlossen“ für Ausführung interessant.

5.1.1 Starten

Der Start einer Prozessinstanz unterscheidet sich funktional stark von den anderen Steuerungsmöglichkeiten, da Prozessinstanzen nur indirekt gestartet werden können. Ein Prozess kann nur durch ein Eingangs-Dokument gestartet werden, das zuvor anhand von Parametern generiert und publiziert werden muss. Dieses Dokument wird auf den Enterprise Service Bus gegeben und bewirkt, dass für alle assoziierten Prozesse ein Trigger

ausgelöst wird. Durch diesen Trigger bemerkt das Prozessausführungssystem das Dokument und startet alle Prozesse, die dieses Dokument als Eingangsdokument haben.

Mit dem Ansatz werden aber von der Prozessausführung für jeden Prozess mit dem publizierten Dokument als Eingang eine Instanz gestartet. Hier wäre ein direkter Start einer Prozessinstanz besser. Da es aber keinen anderen Weg gibt, einen Prozess zu starten, muss dieser Ansatz gewählt werden. Das Problem entsteht dadurch, dass der Instanz-Start asynchron verläuft. Bei einer synchronen Start-Anweisung könnte jede zu startende Prozessinstanz einzeln bestimmt werden. So wird nur das voraussehbare Verhalten des Prozessausführungssystems angenommen und ein Dokument publiziert, welches den Start eines Prozesses auslöst.

In der Regel sind die Eingangsdokumente so gestaltet, dass sie tatsächlich nur für ein Prozessmodell zutreffen. Für das Prozessmodell „Import-Warenverteillager“ gibt es z. B. das Eingangsdokument „Sendungs-Dokument“. Auf diese Weise kann verhindert werden, dass nicht betroffene Prozessinstanzen weiterer Prozesse gestartet werden.

5.1.2 Pausieren, Wiederaufnehmen und Abbrechen

Im Gegensatz zum Prozessstart können die folgenden Prozesssteuerungen direkt auf Prozessinstanzen zugreifen.

Damit eine Prozessinstanz pausiert werden kann, muss sie im Zustand „gestartet“ bzw. „wiederaufgenommen“ sein. Soll die Instanz pausiert werden, ist dazu der eindeutige Bezeichner (Prozessinstanz-ID) der Prozessinstanz notwendig. Die Schnittstelle selbst überwacht den Zustand und pausiert Instanzen nur dann, wenn sie auch laufen. Sollte der Prozess in einem anderen Zustand sein, wird der Zustand nicht geändert. Die Wiederaufnahme einer Prozessinstanz funktioniert analog und nimmt nur Instanzen wieder auf, die zuvor pausiert wurden.

Auf ähnliche Weise wie die Pause funktioniert der Stop der Prozessinstanz. Mit dem Unterschied, dass laufende Instanzen abgebrochen werden. Pausierte Instanzen können wiederaufgenommen werden, wohingegen abgebrochene Instanzen lediglich neu eingesetzt werden können.

Eine Prozessinstanz kann überall und jederzeit pausiert bzw. abgebrochen werden. Falls der pausierte bzw. gestoppte Prozess mit einer Aktivität beschäftigt ist, wird die Aktivität nicht abgebrochen und läuft weiter. Daher wird eine Prozessinstanz effektiv immer erst dann pausiert bzw. abgebrochen, wenn sie einen Sequenzfluss erreicht.

5.1.3 Bewertung

Ansatz	*Änderungs-Klasse*	*Muster*
Prozess-Steuerung	Prozess-Steuerung	Starten, Pausieren, Wiederaufnehmen, Abbrechen
Criteria of Change	*Unterstützung*	*Anmerkungen*
Abstraktionsebene		
Modell		
Instanz	×	
Perspektive		
Funktional	×	Start und Abbrechen
Organisatorisch		
Verhalten	×	Pausieren und Wiederaufnehmen
Informationell	×	Beim Start eines Prozesses wird ein Daten-Objekt angelegt.
Operational		
Eigenschaften		
Ausdehnung	inkremental	
Dauer	temporär	
Unmittelbarkeit	unmittelbar	
Vorwegnahme	unvorhergesehen	

Tabelle 5.1: Flexibilisierungs-Eigenschaften des Ansatzes „Prozess-Steuerung“

Prozessinstanzen können auf die vorgestellte Weise gestartet, pausiert, wiederaufgenommen und gestoppt werden. Eine Prozessinstanz kann durch die Steuerung flexibilisiert werden, beispielsweise wenn sie durch eine Pause vom weiteren Ablauf abgehalten wird und so nachfolgende Aktivitäten nicht ausgeführt werden. In der Tabelle 5.1 sind die Eigenschaften dieses Ansatzes zur Flexibilisierung durch Prozesssteuerung aufgeführt.

Prozessinstanzen können nur auf Instanz-Ebene gesteuert werden. So führt der Start eines Prozesses dazu, dass eine neue Prozessinstanz erstellt wird. Durch Pausieren, Wiederaufnehmen und Abbrechen kann über den Prozess-Zustand das Verhalten der Instanz verändert werden. Neben der Verhaltens-Perspektive muss durch die Neu-Instanziierung des Prozesses auch ein Daten-Objekt angelegt werden, deswegen streift dieser Ansatz beim Prozessstart auch die Daten-Perspektive.

Da dieser Ansatz mit der Instanz eines Prozessmodells arbeitet, sind die Änderungen inkremental. Das bedeutet, dass nur lokale Änderungen in der abgeleiteten Instanz eines bestehenden Prozessmodells durchgeführt werden. Die Steuerung der Prozessinstanz ist

nur von temporärer Dauer und ist unmittelbar und nicht verzögert. Zudem kann mithilfe des Ansatzes eine Prozessinstanz ohne vorherige Vorbereitungen unvorhergesehen gesteuert werden.

5.2 Ad-hoc-Änderungen durch Prozessmodell-Austausch

Der Austausch des Prozessmodells gehört zur Klasse „Flexibilität durch Veränderung“. Mit dem Ad-hoc-Austausch des Modells können funktionale Änderungen an laufenden Prozessinstanzen gemacht werden. Darunter fallen z. B. Änderungen an Aktivitäten oder Sequenzflüssen. Bei diesem Ansatz werden jedoch technisch-bedingt nicht nur die einzelnen Elemente einer einzelnen Prozessinstanz verändert, sondern alle Elemente des entsprechenden Prozessmodells. Ziel dieses Ansatzes ist es, diese technische Einschränkungen der Process Engine zu umgehen und mithilfe des kompletten Prozessmodell-Austauschs einzelne Veränderungen an Prozessinstanzen vornehmen zu können.

webMethods bietet den Austausch eines Prozessmodells als Funktion an. [Sof-2009f][Seite 101] Der Austausch ist nichts anderes, als die Funktionalität, ein bereits aufgespieltes Prozessmodell vor dem erneuten Aufspielen[1] zu validieren und dem System in einer bestimmten Version zur Verfügung zu stellen. Die Prozessausführung von webMethods ermöglicht zudem die Versionierung von Prozessmodellen. Das heißt, der Prozess „Import-Warenverteillager“ kann parallel in beliebig vielen unterschiedlichen Versionen existieren. Die Austausch-Schnittstelle ermöglicht den Austausch eines Prozessmodells einer bestimmten Version, da beim Austausch eine Zielversion angegeben werden kann. Eine Prozessinstanz wird immer für die höchste Version eines Modells gebildet.

Der Austausch des Modells ist interessant, weil durch die Process Engine beim Austausch rückwirkend alle laufenden Instanzen eines Modells in die neue Version migriert[2] werden können. Eine Instanz der Version i wird in dem Moment angepasst, in dem das Prozessmodell der Version i aufgespielt und aktiviert wird. Das heißt, dass es eine Menge von Instanzen der Version 1 und eine Menge von Instanzen der Version 2 geben kann. Wird die Version 1 verändert, betrifft dies nicht die Instanzen der Version 2 und umgekehrt. Dieser Austausch unterscheidet jedoch die betroffenen Instanzen einer Version nicht untereinander, sondern gilt immer für alle Instanzen einer Modell-Version. Flexibilisiert wird hier also auf Prozessmodell-Ebene und nicht auf Prozessinstanz-Ebene. Durch die technische Einschränkung beziehen sich alle Veränderungen immer auf das Prozessmodell einer bestimmten Version.

[1] engl. *Deployment*
[2] Die webMethods-Dokumentation spricht von einem „Upgrade“. [Sof-2009f][Seite 101]

Die Versionierung erlaubt es dem Geschäftsprozessmodellierer, eine aktuelle oder neue Version zu erstellen und eine alte Version beizubehalten. Damit können mehrere Versionen eines Modells parallel zueinander existieren und ablaufen. Da die Flexibilisierung nur temporär sein soll, werden Änderungen nur an einer bestimmten Version des Modells durchgeführt. Eine spätere Version soll von diesen temporären Änderungen unbetroffen sein. Wird eine Flexibilisierung notwendig, gibt es drei Versionen eines Modells: die Aktuelle (i), die Flexibilisierte (i') und die Zukünftige (i+1). Die aktuelle Version ist dabei effektiv dieselbe Version wie die zukünftige. Damit soll bewirkt werden, dass unbetroffene Instanzen, die vor der Flexibilisierung gestartet werden, in der gleichen Form ablaufen wie Instanzen, die nach der Flexibilisierung gestartet werden. Damit diese Variante funktioniert, muss ein Weg gefunden werden, einzelne Prozessinstanzen zu manipulieren, ohne das Prozessmodell zu verändern. Die temporären Änderungen an bestimmten Instanzen sollen also weder alle Instanzen eines Modells noch zukünftige Instanzen dieses Modells betreffen. Dazu wird die aktuellste Modellversion i verändert und als Version i' aufgespielt. Anschließend wird das unveränderte Prozessmodell i als neue Version i+1 erstellt. Neue Instanzen laufen damit weiterhin auf Basis der unveränderten Modellversion (i+1). Bereits laufende Instanzen werden in das Modell i' überführt. Siehe Tabelle 5.2.

Zeitpunkt	*Version*	*Instanzen*	*Prozessmodell*
t-1	i	Abgeschlossene	A
t	i'	Laufende	A (flexibilisiert)
t	i+1	Zukünftige	A

Tabelle 5.2: Zwischenversionen beim Prozess-Deployment

In webMethods werden Instanzen immer von der höchsten Versionsnummer eines Prozessmodells gebildet. Damit eine Flexibilisierung nur die laufenden Instanzen betrifft, wird das Modell der höchsten Version ausgetauscht und durch eine flexibilisierte Variante desselben ersetzt. Um zukünftige Instanzen wieder in der unflexibilisierten Fassung ablaufen zu lassen, wird das zuvor unflexibilisierte Modell mit einer Version i+1 neu publiziert. Auszug 5.1 zeigt diesen Vorgang in Pseudo-Code.

```
oldmodel = getModel(v)
flexModel = flex(oldmodel, newmodel)
deploy(flexModel, v)
deploy(oldmodel, v+1)
```

Auszug 5.1: Austausch und Versionierung des Modells in Pseudo-Code

Sollen auch zukünftige Instanzen flexibilisiert ablaufen, kann darauf verzichtet werden, die alte Version wiederherzustellen. Alle kommenden Instanzen laufen somit wie ihre

Vorgänger auf der flexiblen Version. Diese Variante ist nicht temporär, sondern evolutionär.

Das grundsätzliche Problem dieses Ansatzes ist aber, dass zwar die gewünscht zu verwendende Abstraktionsebene die Instanz-Ebene ist, effektiv aber auf Modell-Ebene gearbeitet wird. Dazu muss ein zusätzlicher Mechanismus auf Modell-Ebene gefunden werden, um laufende Instanzen der Version i' in betroffene und unbetroffene Instanzen zu unterschieden. Um Instanzen desselben flexibilisierten Prozessmodells voneinander zu unterscheiden, müssen spezielle Flexibilitäts-Sequenzflüsse in das flexibilisierte Prozessmodell eingefügt werden. Die bedingten Sequenzflüsse unterscheiden zwischen betroffenen und unbetroffenen Instanzen. Ist eine Instanz unbetroffen, läuft sie den unveränderten Pfad. Ist sie jedoch betroffen, läuft sie den flexibilisierten Pfad. Dazu ist es notwendig, auf Ebene der Sequenzflüsse Prozessinstanzen voneinander zu trennen und auf verschiedene bedingte Sequenzflüsse leiten zu können.

5.2.1 Anwendung nur auf betroffene Instanzen

Da Veränderungen sich immer auf das Prozessmodell und damit auf alle laufenden Instanzen auswirken, muss hier ein Workaround gefunden werden. Dieser Workaround muss in der Lage sein, zusätzliche Pfade für betroffene und unbetroffene Instanzen dem Modell hinzuzufügen, um so betroffene Instanzen flexibilisiert und unbetroffene Instanzen unflexibilisiert ablaufen zu lassen.

Variante 1 Duplizierung des alten Prozessmodells

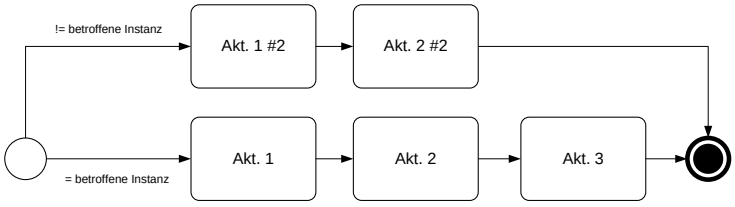

Abbildung 5.1: Variante 1 mit flexibel hinzugefügter Aktivität Akt. 3

Dazu bieten sich zwei Varianten an. Die erste Variante dupliziert den unveränderten Prozesskörper mit allen Steps. Unbetroffene Instanzen durchlaufen den duplizierten Teilprozess, während betroffene Instanzen den veränderten Teilprozess durchlaufen. Siehe dazu Abbildung 5.1 in der die beiden unveränderten Aktivitäten Akt. 1 und Akt. 2 für unbetroffene Instanzen dupliziert werden.

Ein Problem tritt allerdings auf, wenn der Prozessablauf schon einen gewissen Punkt überschritten hat und in einem der Teilprozesse „gefangen“ ist. Mit dieser Variante kann eine Prozessinstanz also nicht zu einem beliebigen Zeitpunkt flexibilisiert werden. Flexibilisiert werden kann demnach nur zu einem Zeitpunkt, der einen Übergang zu einem Teilprozess nach sich zieht. Dies ist bei betroffenen Prozessinstanzen kritisch, die an Stellen des Prozessmodells stehen, die der Flexibilitäts-Verzweigung nachfolgen. Am Beispiel wäre das der Fall, wenn der Prozess durch Hinzufügen flexibilisiert wird und eine unbetroffene Instanz an „Akt. 2“ steht. Obwohl nicht betroffen, würde diese Instanz dann zu „Akt. 3“ übergehen. Die Instanz ist im flexibilisierten Bereich gefangen und kann nicht mehr in den unflexibilisierten Bereich gelangen, da sie eine Pfad-Entscheidung am Anfang ihres Ablaufs verpasst hat.

Variante 2 Duplizierung veränderter Prozesselemente

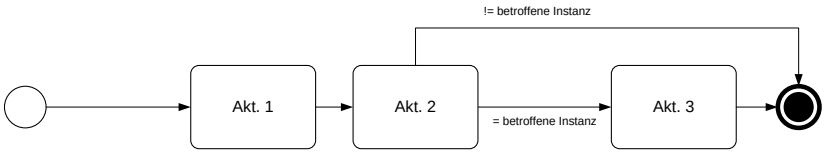

Abbildung 5.2: Variante 2 mit flexibel hinzugefügter Aktivität Akt. 3

Diese Variante vergleicht den unveränderten Prozess mit dem veränderten Prozess und generiert ein flexibilisiertes Prozessmodell, in dem das Delta der beiden Modelle abgelegt ist. Wird beispielsweise eine Aktivität gelöscht, wird im flexibilisierten Prozessmodell ein zusätzlicher Pfad angelegt. Dieser Pfad macht die gelöschte Aktivität für alle unflexibilisierten Instanzen nach wie vor zugänglich und für betroffene Instanzen unzugänglich. Steht die Instanz auf einer Aktivität, die im veränderten Prozess entfernt wird, geht die Instanz nach der vollendeten Aktivität zur nächsten Aktivität über und läuft weiter. Wird eine Aktivität hinzugefügt, erhält die vorangegangene Aktivität einen bedingten Pfad, der die neue Aktivität nur für flexible Prozesse zugänglich macht. Sollte eine Aktivität modifiziert werden, das heißt in ihren Eigenschaften verändert werden, wird sie dupliziert und die unmodifizierte Version für unbetroffene Instanzen und die modifizierte Version für betroffene Instanzen zugänglich gemacht.

Die zweite Variante dupliziert zwar ebenso die Elemente des flexibilisierten und unflexibilisierten Prozesses, geht jedoch wesentlich feiner auf Änderungen des Prozesses ein. Siehe dazu Abbildung 5.2. Flexibilisierte und unflexibilisierte Instanzen können auf diese Weise jederzeit auf ihre zugehörigen Pfade gelangen.

Nun muss eine Möglichkeit gefunden werden, unbetroffene und betroffene Instanzen voneinander zu unterscheiden.

5.2.2 Auswahl des betroffenen Pfades anhand der Instanz

Auch hier bieten sich zwei Möglichkeiten an. Die erste Möglichkeit wäre ein entfernter Service-Aufruf, der auf eine Liste von zu flexibilisierenden Varianten zugreift und dann je nach Prozessinstanz-ID „true“ oder „false“ ausgibt. Das Ergebnis dieses Services entscheidet dann, welcher Pfad eingeschlagen wird. Der Grund, warum diese Möglichkeit vernachlässigt werden sollte, ist der zusätzliche Aufwand, der betrieben werden muss, um an die Pfadvariable zu gelangen. Nach jedem Element des Prozessmodells muss zusätzlich zu den Mehrfach-Pfaden noch eine Abfrage nach der Pfadvariablen eingebaut werden. Die Pfadvariable muss zudem mehrfach in jedem Prozess abgefragt werden.

Die zweite Möglichkeit ist das statische Setzen von Bedingungen. Im Gegensatz zur ersten Möglichkeit wird hier der zu gehende Pfad nicht aufgrund eines Service-Aufrufs ausgewählt, sondern aufgrund einer statischen Bedingung im Sequenzfluss. Da die auf die Weise flexibilisierten Prozessmodelle ohnehin nur eine pseudo-dynamische Abbildung auf statische Prozessmodelle sind, ist eine solche Variante legitim. Diese Variante hängt sämtliche zu flexibilisierenden Instanzen in eine Bedingung der Form `InstanzID == a0...ODER InstanzID == bf...` und schreibt diese statisch in das flexibilisierte Prozessmodell.

Das folgende Beispiel zeigt den bereits bekannten Prozess „Import-Warenverteillager“ (Abbildung 5.3). Für die Instanzen mit den eindeutigen Bezeichnern „a0“ und „bf“ soll der Prozess flexibilisiert werden. Die Sendungen sollen zur Zeitersparnis nicht zwischengelagert werden, sondern direkt zum Kunden transportiert werden. Dazu werden für die zu flexibilisierenden Instanzen die Aktivitäten „Transport Auftrag zum Zwischenlager“ und „Einlagerung im Zwischenlager“ gelöscht (Abbildung 5.4). Alle nicht betroffenen Instanzen sollen trotz Flexibilisierung wie erwartet ablaufen.

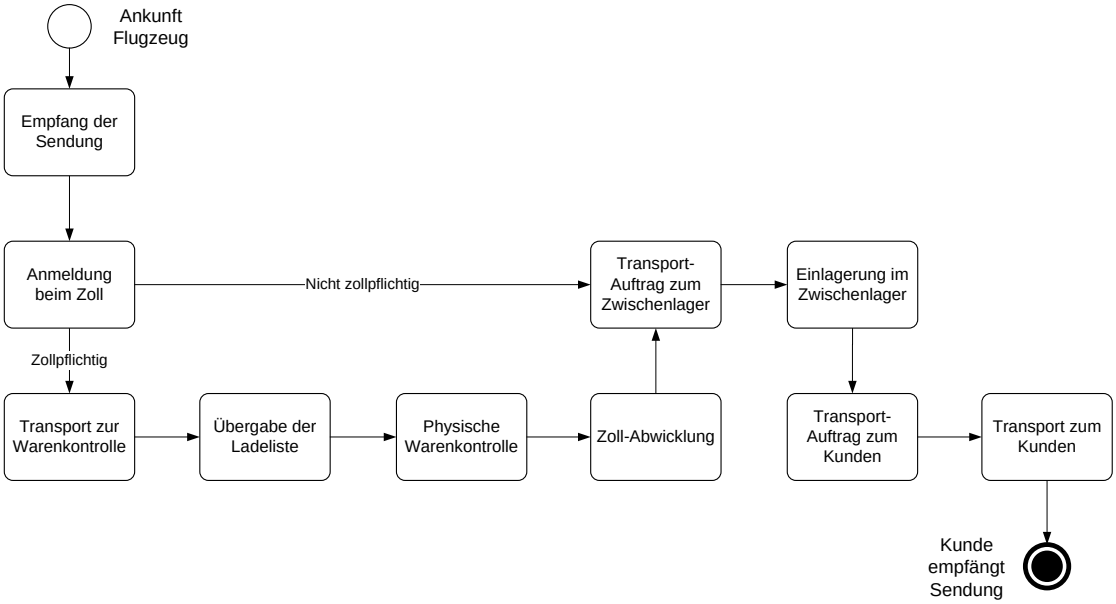

Abbildung 5.3: Unveränderter Prozess „Import-Warenverteillager“

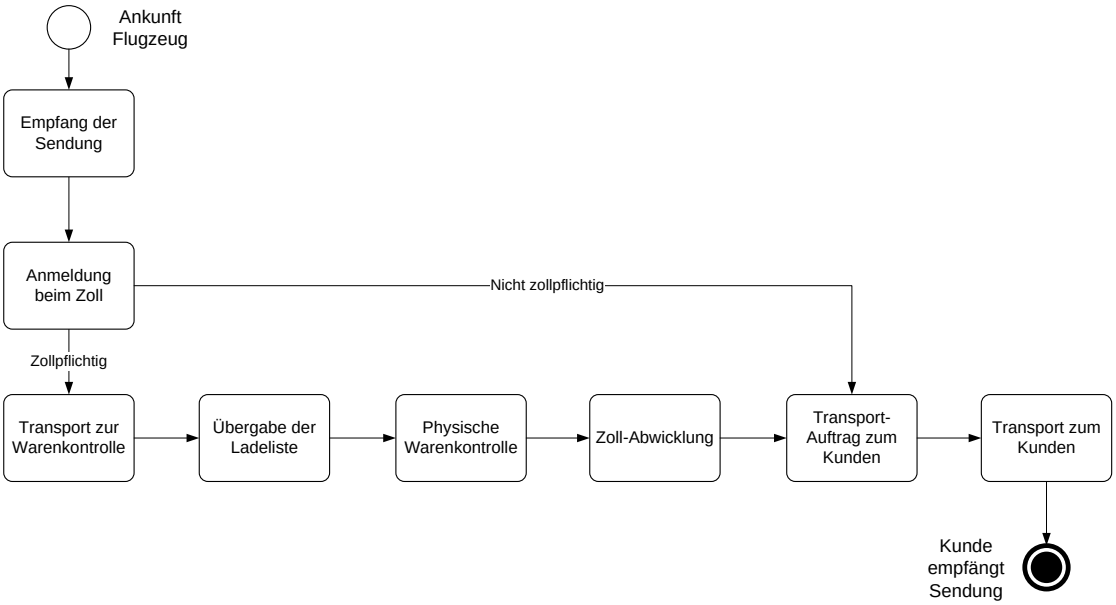

Abbildung 5.4: Veränderter Prozess „Import-Warenverteillager“

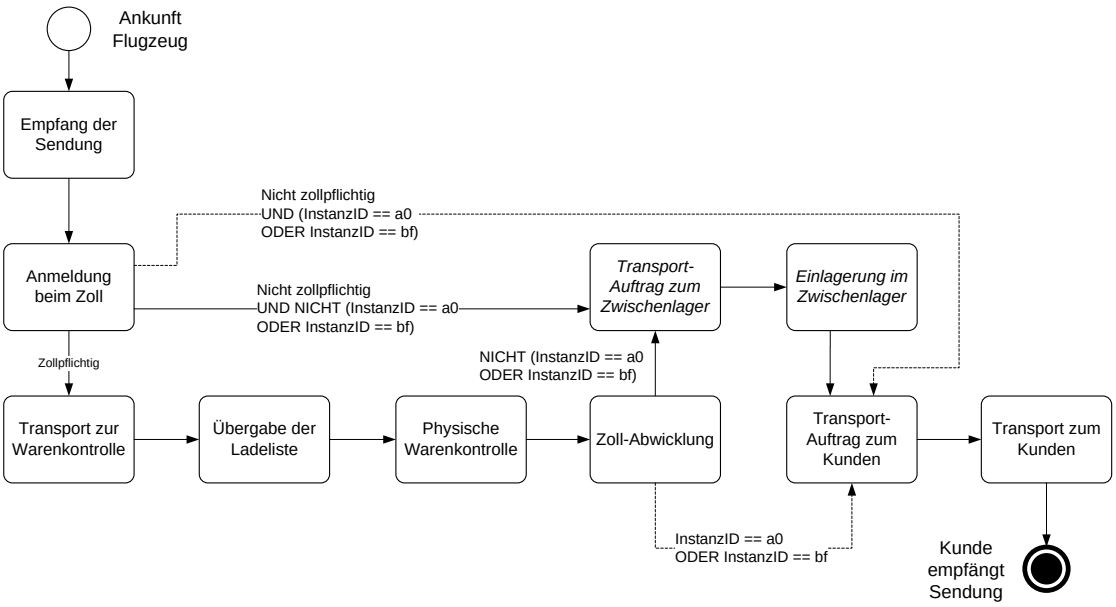

Abbildung 5.5: Flexibilisierter Prozess „Import-Warenverteillager““

Nach der Flexibilisierung enthält das Prozessmodell zusätzliche Sequenzflüsse und Aktivitäten (Abbildung 5.5). Das flexibilisierte Modell enthält alle Änderungen. Zusätzlich sind alle Sequenzflüsse und Aktivitäten aus dem unveränderten Modell enthalten. Damit flexibilisierte und nicht flexibilisierte Instanzen auf demselben Modell laufen können und ihr unterschiedliches Verhalten ausgedrückt werden kann, werden zusätzliche Bedingungen eingefügt. Nicht flexibilisierte Instanzen können, sofern sie keinem der aufgelisteten eindeutigen Instanz-Bezeichner entsprechen, auf das nicht flexibilisierte Modell zugreifen. Flexibilisierte Instanzen hingegen laufen über beschränkte Pfade, die sie zu den Änderungen aus dem veränderten Modell führen.

5.2.3 Meta-Elemente

Bei der Flexibilisierung des Prozessmodells werden Sequenzflüsse und Sequenzfluss-Bedingungen auf einer Meta-Ebene verwendet. Elemente dieser Ebene werden dazu benutzt, um ein Verhalten zu modellieren, das nicht zur funktionalen Ausführung des Prozesses gehört. Vielmehr werden durch sie Mechanismen realisiert, die durch die Flexibilisierung notwendig werden. Offensichtlich wird dieser Unterschied, wenn Sequenzflüsse dem Modell hinzugefügt werden, um betroffene und unbetroffene Prozessinstanzen voneinander zu unterscheiden.

Um das Prozessmodell in seiner Funktionalität korrekt zu belassen, dürfen die Meta-Sequenzflüsse die tatsächlichen Sequenzflüsse funktional nicht verändern. Tatsächlich werden sie zwar verändert, aber in der Ausführung dürfen keine Unterschiede zu einem unveränderten Prozess entstehen. Praktisch heißt das also, dass unbetroffene Prozessinstanzen die auf einem flexibilisierten Modell basieren, funktional genauso ablaufen müssen wie unbetroffene Prozessinstanzen die auf einem unflexibilisierten Modell basieren.

Logische Vereinigung

Im einfachsten Fall verfügt beispielsweise jedes Prozesselement, über eine Aktivität und über genau einen ausgehenden Sequenzfluss. Diesem Sequenzfluss folgt in der Regel ein weiteres Prozesselement nach. Wird eine Aktivität flexibilisiert, muss die im Sequenzfluss nachfolgende Aktivität über eine logische Oder-Verknüpfung verfügen. Da statt bisher nur einem ankommenden Pfad, nun zwei Pfade zur Aktivität führen, müssen die beiden Sequenzflüsse vereinigt werden. Der zusätzlich hinzugefügte Pfad ist Teil der Flexibilisierungs-Meta-Ebene.

In diesem einfachen Fall tritt kein Problem auf, da das nachfolgende Prozesselement lediglich um ein logisches „Oder“ erweitert werden muss. Das Prozesselement akzeptiert also den Eingang beider hinzugefügter Sequenzflüsse. Zu einem Problem kommt es dann, wenn die Aktivität bereits über eine logische Verknüpfung verfügt, wenn z. B. mehr als ein eingehender Sequenzfluss bereits zur Aktivität führt. In diesem Fall verletzt die Meta-Ebene die funktionale Ebene des Prozessmodells. Würde hier eine Oder-Verknüpfung der eingehenden Sequenzflüsse gesetzt werden, würde die bereits bestehende Logik überschrieben werden. Dies würde dazu führen, dass der Prozess seine funktionale Ähnlichkeit verlieren würde. Die Meta-Ebene würde die funktionale Ebene verletzen. Besteht eine logische Verknüpfung im Element und würden damit Meta- und funktionale Ebene miteinander im Konflikt stehen, muss der Konflikt logisch aufgelöst werden. Folgende Aufzählung beschreibt die logische Umformung der eingehenden Sequenzflüsse.

Einfacher Sequenzfluss Das nachfolgende Element verfügt über einen eingehenden Sequenzfluss 1. Da es nur einen eingehenden Sequenzfluss gibt, gibt es keine Vereinigungs-Regel. Die Änderung ist unkritisch. Siehe Abbildung 5.6.
Der Meta-Sequenzfluss 1 wird mit dem bestehenden Sequenzfluss logisch verodert: Sequenzfluss 1 ODER Meta-Sequenzfluss 1.

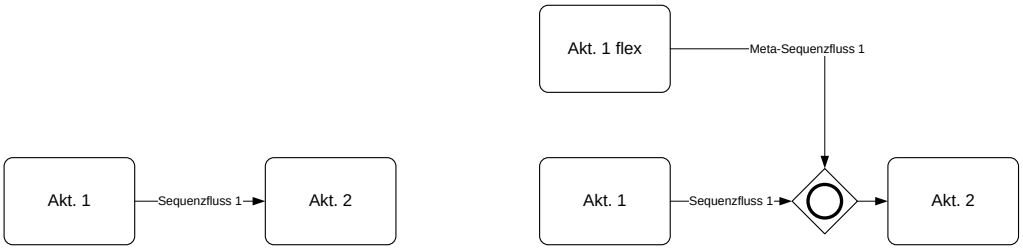

Abbildung 5.6: Einfacher Sequenzfluss

Mehrfacher Sequenzfluss Das nachfolgende Element verfügt bereits über zwei oder mehr eingehende Sequenzflüsse und hat deshalb bereits eine Vereinigungs-Regel definiert. In diesem Fall sei dies UND. Diese Vereinigungs-Regel soll funktional beibehalten und erweitert werden. Siehe Abbildung 5.7.
Sequenzfluss und Meta-Sequenzfluss werden verodert. Diese Meta-Logik wird umklammert, verodert und durch die bestehende Logik erweitert:
(Sequenzfluss 1 ODER Meta-Sequenzfluss 1) UND Sequenzfluss 2.

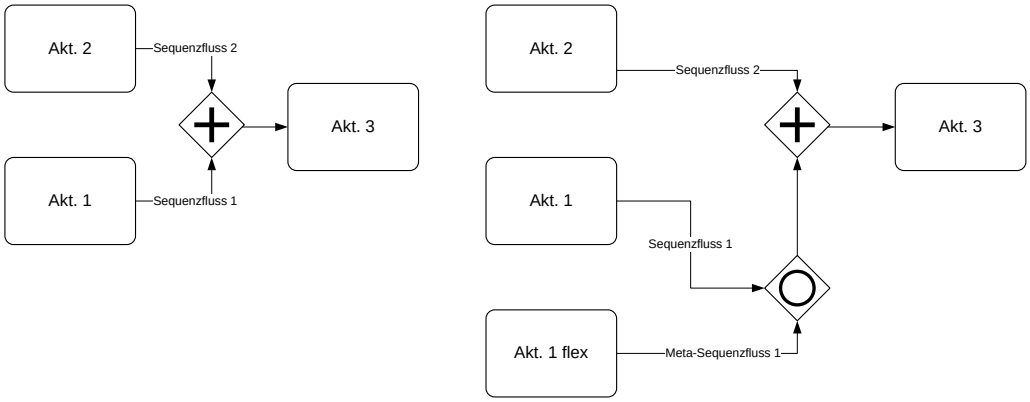

Abbildung 5.7: Mehrfacher Sequenzfluss

Bedingter Sequenzfluss

Sequenzflüsse werden bei dieser Lösung als Teil der Flexibilisierung verwendet. Wenn zum Beispiel eine Aktivität aus dem Modell gelöscht wird, wird der Sequenzfluss zur Aktivität durch eine Bedingung für die flexibilisierte Instanz unzugänglich gemacht. Das führt dazu, dass dem hinführenden Sequenzfluss eine Bedingung hinzugefügt wird. Um hier, die möglicherweise bereits bestehende Logik nicht zu verletzten, müssen die existierenden Bedingungen umgestellt werden. Existiert noch keine Bedingung, kann die Meta-Bedingung ohne Weiteres gesetzt werden. Besteht bereits eine logische Abfrage, muss diese erweitert werden. Zuerst soll dabei überprüft werden, ob die Sequenz berechtigt ist, den Sequenzfluss zu betreten (Meta-Bedingung) und danach sollen die funktionalen Bedingungen überprüft werden (Funktionale Bedingung).

Als ein Beispiel soll die Aktivität „Transport zur Warenkontrolle“ modifiziert werden. Dazu muss der hinführende Sequenzfluss dupliziert und um eine Meta-Sequenzfluss-Bedingung erweitert werden. Besteht bereits ein bedingter Sequenzfluss wie z. B. die Bedingung „Zollpflichtig“, dann wird die Bedingung um (NICHT „Betroffene Instanz“ UND „Zollpflichtig“) für unbetroffene Instanzen und um („Betroffene Instanz“ UND „Zollpflichtig“) für betroffene Instanzen erweitert.

5.2.4 Bewertung

Ansatz	*Änderungs-Klasse*	*Muster*
Ad-hoc-Änderungen durch Prozessmodell-Austausch	Veränderung	Hinzufügen, Löschen, Bewegen, Ersetzen, Vertauschen, Schleife, Parallelisierung, Ein- und Austritt ändern, Parallelisierung, Bedingte Ausführung, Sequenzfluss verändern, Sequenzflussbedingung verändern
Criteria of Change	*Unterstützung*	*Anmerkungen*
Abstraktionsebene		
Modell	×	Technisch bedingt
Instanz	×	
Perspektive		
Funktional		
Organisatorisch		
Verhalten	×	
Informationell		
Operational	×	
Eigenschaften		
Ausdehnung	inkremental	
Dauer	temporär	
Unmittelbarkeit	unmittelbar	
Vorwegnahme	unvorhergesehen	

Tabelle 5.3: Flexibilisierungs-Eigenschaften des Ansatzes „Ad-hoc-Änderungen durch Prozessmodell-Austausch“

Der vorgestellte Ansatz zum Austausch des Prozessmodells gehört zur Klasse „Veränderung“. Dadurch, dass das gesamte Prozessmodell ausgetauscht wird, können alle denkbaren Änderungen am Prozessmodell ausgeführt werden. Die Grenzen sind hier nur durch das Modellierungs-Werkzeug gegeben, mit dem der vorherige Prozess bereits modelliert wurde.

webMethods bietet keine Schnittstelle an, mit der einzelne Prozessinstanzen verändert werden können. Beim Aufspielen werden alle laufenden Prozessinstanzen in die neu aufgespielte Version des Prozessmodells migriert. Allerdings kann der Prozessmodell-Austausch ausgenutzt werden, ein verändertes Instanz-abhängiges Prozessmodell aufzuspielen. Um den Ansatz auf Instanz-Ebene arbeiten zu lassen, werden bedingte Sequenzflüsse so im flexibilisierten Prozessmodell modelliert, dass die Änderungen nur für bestimmte Instanzen spürbar werden.

Da der webMethods Designer verwendet wird, der auch bei der normalen Modellierung zum Einsatz kommt, wird es dem Prozessmodellierer ermöglicht, den Prozess beliebig auf der Verhaltens- und der operationalen Ebene zu verändern. Aktivitäten können hinzugefügt, gelöscht oder im Modell verschoben werden. Auf dieselbe Weise können die Sequenzflüsse umgestellt werden.

Auch dieser Ansatz nimmt inkrementale Veränderungen am Prozessmodell vor, erstellt also kein neues Prozessmodell, da das unveränderte Prozessmodell gültig bleibt. Die Änderungen sind nur temporär, da die Änderungen nur für die laufenden Instanzen gelten und alle zukünftigen Instanzen durch die Versionierung wieder auf dem vorherigen Modell basieren. Sämtliche Änderungen können unmittelbar und ohne Planung durchgeführt werden.

5.3 Step-Resubmit

In webMethods können Steps von Prozessinstanzen zu frei wählbaren Zeitpunkten *resubmittet*[3] werden können. Der Step-Resubmit entspricht in verschiedenen Ausprägungen den Mustern „Neue Aktivitäts-Instanz“ und „Wiederholung“ aus der Klasse „Abweichung im Sequenzfluss“.

Steps können durch einen gezielten Sprung resubmittet werden. Im Folgenden wird statt „Resubmit“ der Begriff „Wiederholung“ verwendet und gleichbedeutend für Steps von Aktivitäten gesprochen. Die Aktivitäten eines Prozesses können laut Dokumentation dann wiederholt werden, wenn sie entweder im Status „fehlerhaft beendet“ oder „erfolgreich beendet“ sind. Im Gegensatz zur Aussage der zugehörigen Dokumentation können Aktivitäten aber auch dann wiederholt werden, wenn sie pausiert sind. [Sof-2009e, Seite 13]

Für die Aktivitäten-Wiederholung kommen also nur Aktivitäten infrage, die zuvor als „wiederholbar“[4] eingestellt wurden. Diese Markierung hat keine tief greifenden Auswirkungen auf den Prozess. Sie bewirkt lediglich, dass verschiedene Prozessinstanz-Parameter wie das Daten-Objekt durch die Ausführungsumgebung des Prozesses mitgeloggt werden. Allerdings würde dieses Logging im größeren Rahmen zu Performance-Problemen führen. [Sof-2009f]

Als Teil dieser Funktionalität kann zudem das Daten-Objekt einzelner Instanzen verändert werden. Davon profitieren die beiden Ansätze „Veränderung der Prozess-Variablen“ im Abschnitt 5.4.1 und „Dynamischer Service-Aufruf“ im Abschnitt 5.4.1.

[3] dt. wieder vorgelegt / wiederholt
[4] engl. resubmittable

5.3.1 Neue Aktivitäts-Instanz

Wird eine Aktivität einer pausierten Instanz wiederholt, wird der Prozess an der Stelle der Aktivität durch eine zusätzliche Instanz der Aktivität neu aufgesetzt. Mit der betroffenen Aktivität werden alle im Prozessablauf folgenden Pfade und Aktivitäten erneut durchlaufen. Die Instanz wird also um eine zusätzliche Sequenzfluss-Iteration erweitert. Eine solche zusätzliche Iteration wird anhand einer fortlaufenden Iterations-Nummer gekennzeichnet. Beginnend ab der Stelle der Wiederholung werden alle nachfolgenden Sequenzflüsse, Aktivitäten und Gateways erneut betreten und ausgeführt. Stellt man sich den Ablauf einer Prozessinstanz als die Übergabe eines Tokens vor, der von Aktivität zu Aktivität gereicht wird, wird für jede neue Instanz ein zusätzlicher Token hinzugefügt.

In Abbildung 5.8 wird das Verhalten veranschaulicht. Die neue Instanz einer Aktivität führt dazu, dass alle nachfolgenden Aktivitäten durch den zusätzlichen Token erneut beschritten werden, unabhängig davon, ob sie bereits vom ersten Token betreten wurden. Anhand der Tabelle 5.4 sieht man, dass zusätzlich zum laufenden Sequenzfluss ein zweiter Sequenzfluss angelegt wird, der sämtliche Aktivitäten erneut durchläuft.

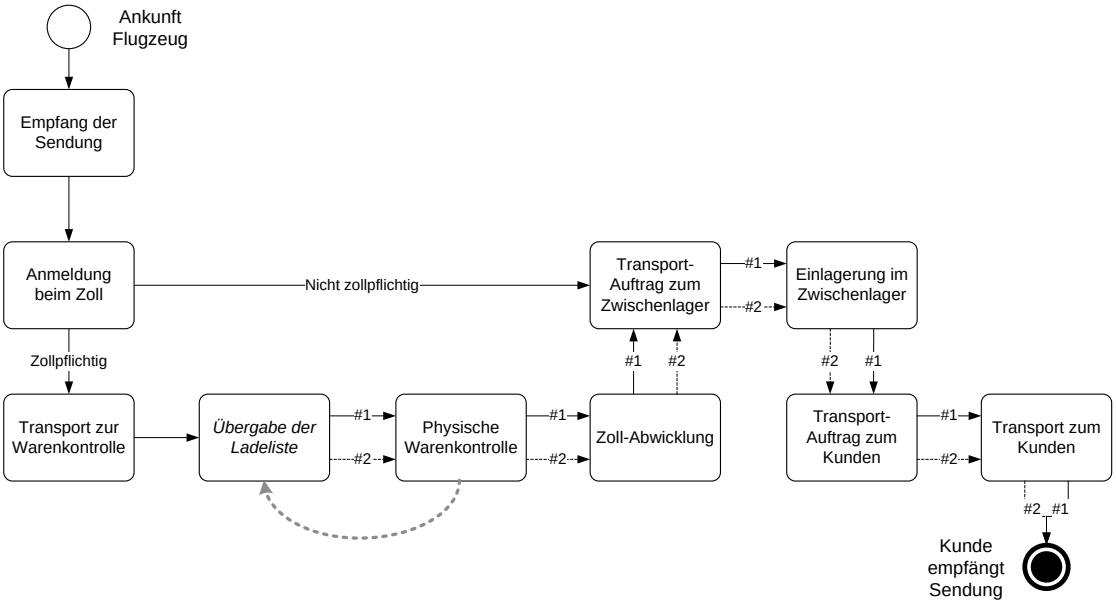

Abbildung 5.8: Wiederholter Prozess „Import-Warenverteillager“

Die Eigenart dieser Variante ist, dass nicht nur die gewählte Aktivität wiederholt wird, sondern alle funktional nachfolgenden Aktivitäten. Dabei wird die vorherige Sequenz nicht beendet oder unterbrochen, sondern läuft weiter. Dadurch entstehen zwei Sequenzen innerhalb eines Prozesses. In vielen Szenarien ergibt dies keinen Sinn, da es in diesen Fällen gleichwertig wäre einen zweiten Prozess zu starten. Am Beispiel des

Warenverteillager-Prozesses soll z. B. nach der Warenkontrolle die Ladeliste erneut übergeben werden. Das heißt, für jede zusätzliche Instanz gibt es ein zusätzliches Token mit nachfolgender Zwischenlagerung und Transport zum Kunden. Eine Wiederholung dieser Ausprägung entspricht dem Muster „Zusätzliche Aktivitäts-Instanz“.

Zeit	*Sequenz 1*	*Sequenz 2*
t	Empfang der Sendung	-
t+1	Anmeldung beim Zoll	-
t+2	Transport zur Warenkontrolle	-
t+3	Übergabe der Ladeliste	-
t+4	Physische Warenkontrolle	*Aktivitäts-Wiederholung*
t+5	Zoll-Abwicklung	Übergabe der Ladeliste
t+6	Transport-Auftrag zum Zwischenlager	Physische Warenkontrolle
t+7	Einlagerung im Zwischenlager	Zoll-Abwicklung
t+8	Transport-Auftrag zum Kunden	Transport-Auftrag
t+9	Transport zum Kunden / *Beendet*	Einlagerung im Zwischenlager
t+10	-	Transport-Auftrag zum Kunden
t+11	-	Transport zum Kunden / *Beendet*

Tabelle 5.4: Zusätzliche Instanz einer Aktivität

5.3.2 Aktivitäts-Wiederholung

Mit einer Erweiterung dieses Ansatzes kann das Muster „Wiederholung“ derselben Klasse erreicht werden. Die Lösung liegt darin, die Prozessinstanz zu stoppen, statt sie nur zu pausieren. In diesem Fall ist der erste Sequenzfluss bereits abgebrochen, wenn der wiederholte Sequenzfluss an der wiederholten Stelle einsetzt. Statt ein zusätzliches Token zu erzeugen, wird das alte Token entfernt und ein neues Token an der Stelle der gewählten Aktivität eingesetzt. Die ausgewählte Aktivität wird wiederholt und die Instanz läuft mit der neuen Sequenzflussiteration weiter. Vgl. Tabelle 5.5 im Unterschied zu Tabelle 5.4. Diese Ausprägung entspricht dem Muster „Wiederholung“.

Zeit	*Sequenz 1*	*Sequenz 2*
t	Empfang der Sendung	-
t+1	Anmeldung beim Zoll	-
t+2	Transport zur Warenkontrolle	-
t+3	Übergabe der Ladeliste	-
t+4	Physische Warenkontrolle	*Aktivitäts-Wiederholung*
t+5	*Gestoppt*	Übergabe der Ladeliste
t+6	-	Physische Warenkontrolle
t+7	-	Zoll-Abwicklung
t+8	-	Transport-Auftrag
t+9	-	Einlagerung im Zwischenlager
t+10	-	Transport-Auftrag zum Kunden
t+11	-	Transport zum Kunden / *Beendet*

Tabelle 5.5: Wiederholung einer Aktivität einer gestoppten Prozessinstanz

5.3.3 Bewertung

Ansatz	*Änderungs-Klasse*	*Muster*
Step-Resubmit	Änderung im Sequenzfluss	Zusätzliche Instanz und Wiederholung
Criteria of Change	*Unterstützung*	*Anmerkungen*
Abstraktionsebene		
Modell		
Instanz	×	
Perspektive		
Funktional		
Organisatorisch		
Verhalten	×	
Informationell	×	technisch bedingt
Operational		
Eigenschaften		
Ausdehnung	inkremental	
Dauer	temporär	
Unmittelbarkeit	unmittelbar	
Vorwegnahme	geplant	Aktivitäten müssen wiederholbar sein

Tabelle 5.6: Flexibilisierungs-Eigenschaften des Ansatzes „Step-Resubmit“

Der vorgestellte Ansatz gehört zur Flexibilitäts-Klasse „Änderung im Sequenzfluss“ und implementiert die Muster „Zusätzliche Instanz“ und „Wiederholung“. Mithilfe des Ansatzes können zusätzliche Instanzen von Aktivitäten gebildet werden, die dann innerhalb einer Instanz eine eigene Sequenz erhalten und so das Muster „Zusätzliche Aktivitäts-Instanz“ ermöglichen. Wird der vorherige Sequenzfluss hingegen abgebrochen, wird die Aktivität wiederholt und der neue Sequenzfluss ersetzt den Alten. Diese Ausprägung ermöglicht das Muster „Wiederholung“.

Änderungs-Subjekte dieses Ansatzes sind das Verhalten der Prozessinstanz und das Daten-Objekt. Das Daten-Objekt ist Teil der genutzten Funktionalität und kann bei der Flexibilisierung ignoriert werden. Das Verhalten des Prozesses wird dahin gehend verändert, dass eine zusätzliche Sequenz zu der bereits bestehenden Sequenz erstellt wird bzw. die Laufende abgebrochen und eine Neue eingesetzt wird.

Wie alle bisherigen Ansätze ist dieser Ansatz inkremental, von temporärer Dauer und kann unmittelbar und unvorhergesehen durchgeführt werden. Eine Einschränkung gilt hier bei der unvorhergesehenen Durchführung, da die Aktivitäten des Prozessmodells

als wiederholbar markiert sein müssen. Dadurch muss eine Flexibilisierung dieser Art geplant sein.

5.4 Dynamisches Daten-Objekt

Der Ansatz „Dynamisches Daten-Objekt“ basiert auf dem Resubmit eines Steps. Das Daten-Objekt einer Instanz kann immer dann verändert werden, wenn ein Prozessschritt durch eine zusätzliche Aktivitäts-Instanz neu eingesetzt wird. Diese Funktionalität wurde im vorherigen Abschnitt vorgestellt. Bei diesem Ansatz wird das Daten-Objekt zur Laufzeit einer Instanz verändert, um ein verändertes Verhalten zu bewirken. Dieser Ansatz ist Teil der Klasse Veränderung. Er bietet die technische Grundlage für die beiden Ansätze „Veränderung der Prozess-Variablen“ und „Dynamischer Service-Aufruf“. Aufgrund der starken technischen Ähnlichkeit werden die beiden Ansätze gemeinsam vorgestellt.

Das Daten-Objekt von webMethods ist ein Teil einer globalen „Pipeline“. Die Pipeline steht im Gegensatz zu anderen BPM-Implementierungen, bei denen ein Daten-Objekt über Daten-Assoziationen von Aktivität zu Aktivität gereicht wird. Dadurch ist es relativ einfach, auf die Daten des globalen Daten-Objekts zuzugreifen und zu verändern.

5.4.1 Veränderung der Prozess-Variablen

Mithilfe dieses Ansatzes kann eine zum Prozessbeginn definierte Variable verändert werden. Dazu wird die Prozessinstanz an der Stelle ihrer aktuellen Ausführung mit veränderten Werten neu eingesetzt. Der übrige Ablauf der Instanz erfolgt dann auf Basis der veränderten Werte. Ein Beispiel hierfür ist, dass die Adresse eines Kunden während des Ablaufs des Warenverteillager-Prozesses geändert werden soll. Auf diese Weise kann die Sendung an eine andere Adresse versendet werden als ursprünglich vorgesehen.

5.4.2 Dynamischer Service-Aufruf

Beim dynamischen Service-Aufruf wird das Daten-Objekt so verändert, dass Referenzen auf Services flexibilisiert werden können. Dazu müssen Services zunächst dynamisch referenziert werden. Services können als Implementierung einer Aktivität aufgerufen werden. Welcher Service aufgerufen wird, wird zur Design-Zeit ausgewählt. Üblicherweise wird der gewünschte Service mitsamt seiner Parameter statisch modelliert, z. B. ruft die Aktivität „Transport-Auftrag zum Zwischenlager“ den Service „Transport-Service“ auf.

Ein Service muss nicht direkt durch die Aktivität aufgerufen werden, sondern kann auch indirekt über einen Invoke-Service gestartet werden. Der Invoke-Service ist ein Service, der den eigentlichen Service-Aufruf kapselt und anderen Anwendungen zur Verfügung stellt. Wird dem Invoke-Service die Service-Referenz übergeben, bietet das den Vorteil, dass ein Service nicht statisch modelliert werden muss, sondern als dynamischer Parameter für den Invoke-Service definiert werden kann.

Die dynamische Referenz zum Service ist in Form einer Variable im Daten-Objekt abgelegt. Siehe Abbildung 5.9. Der Service wird zur Design-Zeit, statt ihn direkt aufzurufen, anhand der Variablen `reference` referenziert und vom Invoke-Service aufgerufen. Die `reference`-Variable wird, z. B. durch einen Standardwert, zum Start der Instanz definiert. Der Service-Parameter `data` enthält das Daten-Objekt der Instanz, das dem Service übergeben wird.

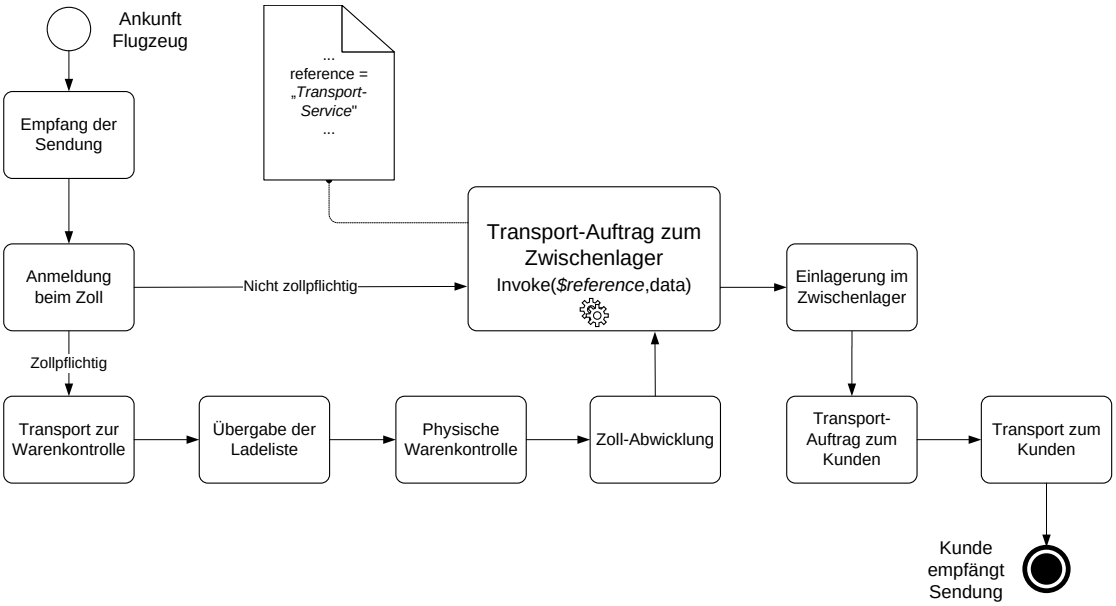

Abbildung 5.9: Dynamische Service Referenz beim Prozess „Import-Warenverteillager“

Wird eine Instanz flexibilisiert, wird das Feld mit der namentlichen Referenz zum Ziel-Service im Datenobjekt verändert. Sollen mehrere Service-Aufrufe in einem Prozessmodell dynamisch sein, muss für jeden dieser Aufrufe ein eigenes Feld im Datenobjekt definiert sein.

5.4.3 Bewertung

Ansatz	*Änderungs-Klasse*	*Muster*
Veränderung der Prozess-Variablen	Veränderung	Daten-Objekt verändern
Criteria of Change	*Unterstützung*	*Anmerkungen*
Abstraktionsebene		
Modell		
Instanz	×	
Perspektive		
Funktional		
Organisatorisch		
Verhalten	×	technisch bedingt
Informationell	×	
Operational		
Eigenschaften		
Ausdehnung	inkremental	
Dauer	temporär	
Unmittelbarkeit	umittelbar	
Vorwegnahme	geplant	Die Pipeline des Prozesses muss geloggt werden

Tabelle 5.7: Flexibilisierungs-Eigenschaften des Ansatzes „Veränderung der Prozess-Variablen“

Ansatz	*Änderungs-Klasse*	*Muster*
Dynamischer Service-Aufruf	Externalisierung	Dynamische Service-Referenz
Criteria of Change	*Unterstützung*	*Anmerkungen*
Abstraktionsebene		
Modell		
Instanz	×	
Perspektive		
Funktional		
Organisatorisch		
Verhalten	×	technisch bedingt
Informationell	×	Referenz im Daten-Objekt
Operational	×	
Eigenschaften		
Ausdehnung	inkremental	
Dauer	temporär	
Unmittelbarkeit	unmittelbar	
Vorwegnahme	geplant	Services müssen dynamisch referenziert sein und die Pipeline des Prozesses mitgeloggt werden

Tabelle 5.8: Flexibilisierungs-Eigenschaften des Ansatzes „Dynamischer Service-Aufruf“

Bedingt durch die technische Verwandtschaft der beiden Ansätze werden die Flexibilisierungs-Eigenschaften gemeinsam bewertet. Tabelle 5.8 zeigt die Flexibilisierungs-Eigenschaften für den dynamischen Service Aufruf und Tabelle 5.7 für Änderungen am Datenobjekt.

Beide Ansätze flexibilisieren auf Instanz-Ebene, da nur Instanz-spezifische Daten verändert werden. Weil bei der Aktivitäts-Wiederholung für Änderungen auf der Daten-Perspektive auch der Sequenzfluss verändert wird, betreffen beide Flexibilisierungs-Ansätze zwangsläufig die Verhaltens-Perspektive. Weil die Basis-Funktionalität dieses Ansatzes an die Wiederholung einer Aktivität geknüpft ist, muss bei jeder Flexibilisierung eine Aktivität wiederholt werden. Um die Prozessinstanz nicht auch auf der Verhaltens-Ebene zu verändern, dürfte nur die aktuell laufende Aktivität wiederholt werden. Dazu müsste diese bestimmt und verhindert werden, dass die Aktivität erneut gestartet wird. Weil dies durch den Step-Resubmit nicht angeboten wird, wird bei der Daten-Objekt-Flexibilisierung auch immer das Verhalten der Instanz geändert.

Beim Ansatz „Dynamischer Service Aufruf“ wird die Instanz durch den unterschiedlichen aufgerufenen Service zudem auf operationaler Ebene flexibilisiert.

Die Ausdehnung beider Flexibilisierungs-Ansätze ist inkrementa l, von temporärer Dauer und Änderungen werden unmittelbar durchgeführt. Da die Pipeline des Prozesses in jedem Fall mitgeloggt werden muss und Services schon zur Modellierung dynamisch referenziert werden müssen, ist die Flexibilisierung geplant.

5.5 Service-Implementierung

Bei dem Ansatz „Flexible Service-Implementierung“ wird die zu flexibilisierende Funktionalität aus dem Prozess heraus in einen Service verlagert. Der Ansatz gehört zur Klasse Externalisierung, weil der aufgerufene Service selbst entscheidet, welches Verhalten ausgeführt wird.

In webMethods BPM können Aktivitäten durch sogenannte Integration-Server-Services (IS-Services) implementiert werden, die im Integration Server abgelegt sind. Die Eingangsparameter der Services sind Teile des Daten-Objekts (*Pipeline*), deren Daten auf die Parameter des Services abgebildet werden. Über das übergebene Daten-Objekt kann er aufgrund des eindeutigen Instanz-Bezeichners die angebotene Funktionalität auswählen. Zusätzlich sind Instanz-relevante Information wie der eindeutige Bezeichner der Instanz oder der Name des entsprechenden Prozessmodells im Daten-Objekt abgelegt.

Im Beispiel in Abbildung 5.10 soll zur Übergabe der Ladelisten, für die Instanz mit dem eindeutigen Bezeichner „ab“ eine E-Mail zum Zoll versendet werden, anstatt wie

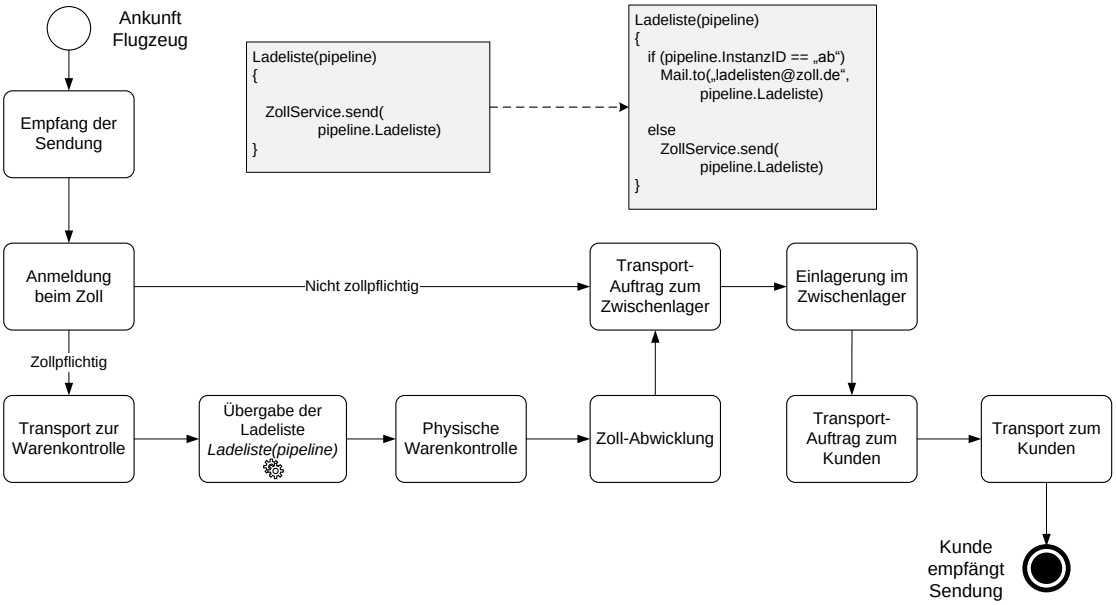

Abbildung 5.10: Externer Service beim Prozess „Import-Warenverteillager“

üblich den Ladelisten-Service des Zolls aufzurufen. Die Abbildung zeigt die ursprüngliche Implementierung des Service „Ladeliste“. Der Service bezieht die Ladeliste aus dem Daten-Objekt und gibt sie an den entfernten Zoll-Service weiter. In der flexibilisierten Service-Version wird der eindeutige Bezeichner des Daten-Objekts der aufrufenden Instanz überprüft. Entspricht dieser dem Wert „ab“, wird eine E-Mail gesandt. Ist dies nicht der Fall, wird das übliche nicht flexibilisierte Verhalten aufgerufen.

5.5.1 Bewertung

Ansatz	*Änderungs-Klasse*	*Muster*
Service-Implementierung	Externalisierung	Externe Implementierung
Criteria of Change	*Unterstützung*	*Anmerkungen*
Abstraktionsebene		
Modell		
Instanz	×	
Perspektive		
Funktional		
Organisatorisch		
Verhalten		
Informationell		
Operational	×	
Eigenschaften		
Ausdehnung	inkremental	
Dauer	temporär	
Unmittelbarkeit	unmittelbar	
Vorwegnahme	unvorhergesehen	

Tabelle 5.9: Flexibilisierungs-Eigenschaften des Ansatzes „Service-Implementierung“

Der Ansatz nimmt eine Sonderrolle ein, da die Instanzen auf operationaler Ebene außerhalb der Prozessausführung flexibilisiert werden. Das heißt, die Prozessinstanz selbst wird nicht flexibilisiert, sondern die entfernte Funktionalität, auf die sie aufbaut.

Die Änderungen, die durch diesen Ansatz durchgeführt werden, sind inkrementa l. Sie manipulieren zwar nicht die Prozessinstanzen selbst, nehmen aber Änderungen an den Services vor, durch die der Prozess operational implementiert ist. Werden die Änderungen an der Implementierung des Services nicht rückgängig gemacht, bleiben sie bis zu einer erneuten Änderung bestehen. Die Flexibilisierung selbst ist nur temporär, da sie in Abhängigkeit eines eindeutigen Instanzbezeichners ablaufen, der nur einmal und zu einem Zeitpunkt auftreten kann. Zukünftige und unbetroffene Instanzen werden nicht flexibilisiert. Da bereits laufende Instanzen von der Flexibilisierung eines externen Services betroffen sind, ist die Flexibilisierung unmittelbar. Wie beim vorherigen Ansatz sind Änderungen geplant, da Services dynamisch modelliert und Aktivitäten wiederholbar sein müssen.

5.6 Zusammenfassung der Flexibilitäts-Muster

Bevor die Flexibilitäts-Muster im nächsten Kapitel implementiert werden, werden sie an dieser Stelle nochmals zusammengefasst. Die Tabelle 5.10 zeigt die vier Flexibilitäts-Klassen und die identifizierten Flexibilitäts-Muster. Die dritte Spalte zeigt an, ob einer der Ansätze eine Möglichkeit bietet, das Muster zu implementieren. Ist dies der Fall, ist der Ansatz in der nachfolgenden Spalte aufgeführt.

Klasse	*Muster*		*Ansatz*
Prozess-Steuerung	Starten	×	Prozess-Steuerung
	Abbrechen	×	
	Pausieren	×	
	Wiederaufnehmen	×	
Sequenzfluss	Rückgängig	-	Step-Resubmit
	Wiederholen	×	
	Überspringen	-	
	Sprung	-	
Externalisierung	Zusätzliche Instanz	×	Step-Resubmit
	Service-Referenz	×	Dynamisches Daten-Objekt
	Flexibler Service	×	*Flexible Service-Implementierung*
Veränderung	Daten-Objekt	×	Dynamisches Daten-Objekt
	Hinzufügen	×	Prozessmodell-Austausch
	Löschen	×	
	Bewegen	×	
	Ersetzen	×	
	Vertauschen	×	
	Schleife	×	
	Parallelisierung	×	
	Ein- und Austritt	×	
	Bedingte Ausführung	×	
	Sequenzflussbedingung	×	
	Sequenzfluss	×	

Tabelle 5.10: Lösungsansätze als Implementierung der Flexibilitäts-Muster

Die Klasse Prozess-Steuerung ist vollständig durch den gleichnamigen Ansatz abgedeckt. Dies liegt zum einen daran, dass die Steuerung von Prozessen trivial ist, zum Anderen, dass der Ansatz maßgeblich die zugehörige Klasse bei deren Definition beeinflusst hat.

In der Klasse „Sequenzfluss“ bietet sich ein anderes Bild. Nur für die Muster „Wiederholung“ und „Zusätzliche Instanz“ konnte ein geeigneter Ansatz für die Implementierung gefunden werden, da willkürliche Sprünge im Sequenzfluss einer Instanz in webMethods nicht vorgesehen sind.

Der „Step-Resubmit“-Ansatz bildet die technische Grundlage für gleich vier Ansätze aus drei verschiedenen Flexibilitäts-Klassen. So kann mithilfe seiner Funktionalität Aktivitäten wiederholt und eine zusätzliche Aktivitäts-Instanz erstellt werden. Darüber hinaus

profitiert das Muster „Service-Referenz“ aus der Klasse „Externalisierung“ und das Muster „Daten-Objekt“ aus der Klasse Veränderung von der Nebenerscheinung, dass beim Step-Resubmit auch das Daten-Objekt der Instanz verändert werden kann.

Das Muster „Flexibler Service“ ist als Einziges nicht abhängig von den Möglichkeiten der Ausführungsumgebung. Ein flexibler Service kann lokal oder entfernt implementiert sein, da er nicht Teil des eigentlichen Prozesses ist. Daher wird die Implementierung dem Anbieter des Prozesses überlassen und das Muster nicht im Prototyp implementiert.

Die Muster in der Klasse „Veränderung“ können alle bis auf das bereits abgedeckte Muster „Daten-Objekt“ mit dem Ansatz „Prozessmodell-Austausch“ durchgeführt werden. Flexibilisiert wird über ein eigens erstelltes Prozessmodell, das ein Delta aus dem alten und dem veränderten Prozessmodell darstellt. Dadurch können alle Möglichkeiten ausgenutzt werden, die die Modellierung bereitstellt. Sofern ein Änderungs-Muster durch modelliert werden kann, kann es auch angewandt werden.

Im folgenden Kapitel werden die Ansätze im Prozessmanipulator-Prototyp implementiert.

Kapitel 6

Prototyp

In diesem Kapitel wird der Prototyp des Prozessmanipulators entwickelt. Der Prozessmanipulator soll in der Lage sein, über eine Schnittstelle aus den implementierten Mustern zur Prozessflexibilisierung auszuwählen. Dazu werden in den folgenden Abschnitten die Funktionen „Start einer neuen Prozessinstanz“, „Prozesssteuerung laufender Instanzen“, „Prozessmodell-Austausch“, „Wiederholung von Aktivitäten“ und „Änderungen am Datenobjekt“ implementiert und deren Einschränkungen gezeigt. Das nachfolgende Kapitel „Integration des Prototyps“ beschreibt die Schnittstelle des Prototyps und schließt mit einer Bewertung der Implementierung.

Der im Rahmen dieser Arbeit entstandene Prototyp ist eine Implementierung des Prozessmanipulators, der folgende Flexibilitäts-Muster ermöglicht: Starten, Pausieren, Wiederaufnehmen, Abbrechen (Prozess-Steuerung), Hinzufügen, Löschen, Bewegen, Ersetzen, Vertauschen, Schleife, Parallelisierung, Ein- und Austritt ändern, Parallelisierung, Bedingte Ausführung, Sequenzfluss verändern, Sequenzflussbedingung verändern, Daten-Objekt verändern (Veränderung), Dynamischer Service-Aufruf (Externalisierung), Wiederholung und Zusätzliche Aktivitäts-Instanz (Abweichung im Sequenzfluss).

6.1 Implementierung

Der Integration Server bietet die Möglichkeit, Services auf zwei Arten zu implementieren. Zum einen durch Flow-Services, die durch eine grafische Sprache implementiert werden können und durch Java IS-Services, die in Java implementiert werden. Der Prozessmanipulator ist als ein Java-IS-Service implementiert. Services können in beliebig verschachtelten Paketen innerhalb des Integration Servers abgelegt werden. Die Services werden im Software AG Designer entwickelt und im Integration Server gespeichert und ausgeführt. Aufgerufen werden können Services durch Prozess-Aktivitäten oder durch JMS-Trigger wie im Falle des Prozessmanipulators.

```
public final class processManipulator_SVC
{
    public static final void processManipulator(IData pipeline)
           throws ServiceException {
           ...
           pm_ProcessManipulator(pipeline);
    }

    // --- <<IS-BEGIN-SHARED-SOURCE-AREA>> ---
    public static IData pm_ProcessManipulator(IData pipeline)
    {
           ...
    }
    ...
    // --- <<IS-END-SHARED-SOURCE-AREA>> ---
}
```

Auszug 6.1: Signatur des Java-IS-Services Prozessmanipulator

Ein Java-IS-Service basiert auf einer unveränderlichen Java-Klasse mit fester Signatur und nur einer Funktion (siehe Auszug 6.1). Der einzige Parameter der Funktion ist ein IData-Objekt, das eine proprietäre Daten-Schnittstelle darstellt. Die Rückgabe-Werte werden über dasselbe Objekt zurückgegeben. Die Funktion wird beim Start des Services durch den IS aufgerufen. Nach der Haupt-Funktion folgt eine „Shared Source Code Area“. Innerhalb dieses Bereiches können beliebig viele Funktionen frei implementiert werden, die zusätzlich mit allen anderen Services innerhalb des Paketes geteilt werden. Die Markierung für geteilten Code wird ebenso für die Klassen-Signatur durch die Entwicklungsumgebung geschützt und kann nicht verändert werden. Die Funktion `process_Manipulator` dient als Eingangs-Funktion der Service-Klasse `processManipulator_SVC` und ruft die Funktion `pm_Processmanipulator` in der „Shared Source Code Area“ auf, die die eigentliche Logik enthält.

6.1.1 Funktionalität

Bedingt durch die Beschränkung auf eine Klasse ist der Prozessmanipulator funktional programmiert. Damit ein gewisser Grad an Übersicht gegeben ist, sind alle implementierten Funktionen nach einem Namespace-Konzept untergliedert. Funktionen mit dem Präfix „pm“ implementieren die konkreten Flexibilisierungs-Ansätze. Die „tool“-Funktionen abstrahieren die API der Prozessausführung. Das ist nötig, weil der API kein durchgängiges Konzept zugrunde liegt. Beispielsweise werden Prozessmodelle auf unterschiedliche Weise adressiert. Zudem ist der Code für einen Service-Aufruf sehr aufwendig. Mithilfe der „tool“-Funktionen wird dieser Aufwand minimiert.

Da außerdem die Implementierung des Prozessmodell-Austausch-Ansatzes sehr komplex ist, sind die Funktionen für diesen Ansatz mit „flex“ markiert. Die „tool“- und „flex“-Funktionen sind, sofern nicht in diesem Kapitel erklärt, im Anhang auf Seite 117 dokumentiert.

Methode	*Funktion*	*Abschnitt*
`pm_Processmanipulator`	Prozessmanipulator	6.2
`pm_List`	Ausgabe der Instanz-Daten	6.3
`pm_Start`	Start einer neuen Prozessinstanz	6.4
`pm_ControlFlow`	Prozesssteuerung laufender Instanzen	6.5
`pm_ReplaceModel`	Prozessmodell-Austausch	6.6
`pm_Redo`	Wiederholung von Aktivitäten	6.7
`pm_NewActivityInstance`	Zusätzliche Aktivitäts-Instanzen	”
`pm_UpdateDataObject`	Änderungen am Datenobjekt	6.8

Tabelle 6.1: Prozessmanipulator-Funktionen

Tabelle 6.1 zeigt die „pm“-Funktionen für die Implementierung der Flexibilisierungs-Ansätze und in welchem der folgenden Abschnitte sie behandelt werden.

6.2 Prozessmanipulator

Die Funktion `pm_Processmanipulator` implementiert eine Controller-Klasse zum Aufrufen der implementierten Ansätze und zum Übergeben der Parameter. Der Prozessmanipulator benötigt die Parameter Aktion (`action`), Prozessmodell (`processtype`), Lösung (`solution`) und Bedingung (`condition`).

Anhand des „Aktion“-Parameters wird die gewünschte Flexibilisierungs-Funktionalität ausgewählt. Der Parameter „Prozessmodell“ verweist auf den eindeutigen Bezeichner des betroffenen Prozessmodells. Der Bezeichner ist wichtig, da die API der Prozessausführung auf das Prozessmodell angewiesen ist, um auf Prozessinstanzen zuzugreifen. Die beiden Parameter „Lösung“ und „Bedingung“ sind Mengen mit Lösungen und Bedingungen für die Prozessflexibilisierung. Mithilfe der Lösung werden notwendige Parameter an die Prozessflexibilisierungs-Funktionen übergeben. Über die Bedingungen können Kriterien spezifiziert werden, die der Auswahl der betroffenen Prozessinstanzen dienen. Die Parameter für den Prozessmanipulator werden innerhalb des Prozessmanipulations-Ereignisses übergeben. Die Aufgabe der Funktion `pm_Processmanipulator` ist es, die Parameter in eine Form zu bringen, die von den Flexibilisierungs-Funktionen interpretiert werden können. Die Funktionen `pm_List`, `pm_Start`, `pm_ControlFlow`, `pm_ReplaceModel`,

`pm_Redo`, `pm_NewActivityInstance` und `pm_UpdateDataObject` teilen folgende Signatur:

```
public static void pm_*(String action, String processtype,
Map<String, String> solutions, Map<String, String> conditions)
```

Die Strings `action` und `processtype` stehen für die auszuführende Aktion und den Prozessmodell-Bezeichner. Die beiden Maps[1] `solutions` und `conditions` können mehrgliedrige Lösungen bzw. Bedingungen darstellen. Gibt es für eine Flexibilisierungs-Funktion nur einen Lösungs- bzw. Bedingungs-Wert, so wird dies durch den leeren Key „“ dargestellt.

Über den `processtype` wird die Änderung auf ein bestimmtes Prozessmodell beschränkt. Prozessmodell-übergreifende Änderungen sind nicht vorgesehen. Der `processtype` bildet den ersten Filter über die Menge der Prozessinstanzen in der Prozessausführung. Die `condition` ist ein zusätzlicher und feinerer Filter. Anhand der `condition` wird die Menge der Prozessinstanzen eingeschränkt, auf die die Flexibilitäts-Funktion Zugriff hat. Mögliche Bedingungen sind z. B. der Zustand der Instanz oder die Prozessvariablen der Instanz. Die Abfrage konkreter Prozessinstanzen, z. B. über ihren eindeutigen Prozessinstanzbezeichner, ist nicht vorgesehen, da die Event-Engine keinen direkten Zugriff auf den Zustand der Prozessausführung hat. Die Abfrage ist also deskriptiv, d. h. über die Prozessdaten oder den Prozesszustand. Die `condition` und der `processtype` sind logisch konjugiert. Die gestellten Bedingungen werden nur auf Instanzen des angegebenen `processtypes` angewandt.

Welche Parameter für die jeweilige Funktion benötigt werden, wird in den folgenden Abschnitten dargestellt. Wie die Eingangsparameter durch das komplexe Ereignis Prozessmanipulation bereitgestellt werden, ist im Rahmen der Prototyp-Integration im nächsten Kapitel beschrieben.

6.3 Auswahl betroffener Prozessinstanzen

In diesem Abschnitt wird die Auswahl der betroffenen Prozessinstanzen mithilfe der `pm_List`-Funktion beschrieben. `pm_List` ist eine Funktion zum Ausgeben von Prozessdaten betroffener Instanzen und dient nicht der Flexibilisierung, sondern lediglich der Veranschaulichung. Die Funktion ist auf die gleiche Weise implementiert wie die restlichen Flexibilisierungs-Funktionen, da die Art wie Bedingungen auf Prozessinstanzen

[1] Ein Key-Value-Daten-Objekt, bei dem es für jeden Schlüssel genau einen Wert gibt.

angewandt werden bei allen Funktionen gleich ist. Dieser Abschnitt zeigt, wie Prozessinstanzen aufgrund ihres Zustands und der Werte ihres Daten-Objekts gefiltert werden können. Beide Bedingungen sind optional.

Parameter	*Wert*
`action`	„List“
`processtype`	Modellname
`solutions`	-
`conditions`	Werte des Daten-Objekts der Instanzen und/oder Instanz-Zustand

Tabelle 6.2: Parameter beim Ausgeben einer Prozessinstanz

```
private static void pm_List(String action, String processtype,
    Map<String, String> solutions, Map<String, String> conditions)
{
    List<IData> instances =
     tool_getInstancesFromCondition (processtype, conditions);

    for (IData instance : instances)
    {
        tool_listInstance(instance);
    }
}
```

Auszug 6.2: Implementierung von `pm_List`

Welche Instanzen für die Ausgabe infrage kommen, entscheidet die Funktion `tool_getInstancesFromCondition` aufgrund des `processtypes` und der übergebenen `conditions`. Die Instanzen werden zuerst implizit nach ihrem Prozessmodell gefiltert, da eine Menge von Instanzen nur anhand ihres Prozessmodells abgefragt werden kann. Die Abfrage wird durch den API-Service `getInstanceList`[2], der alle Instanzen eines Typs zurückgibt, ermöglicht. Die ermittelten Instanzen werden anschließend nach ihrem Zustand und anhand der Werte ihres Daten-Objekts gefiltert. Dabei wird jeder Wert aus den `conditions` mit jedem Wert des Daten-Objekts der Instanz verglichen. Sind die Bedingungen erfüllt, wird die Instanz der Rückgabe-Liste hinzugefügt. Sollten weder der Zustand noch Werte für das Daten-Objekt definiert sein, werden alle Instanzen eines Prozessmodells zurückgeliefert. Dazu zählen auch abgeschlossene, abgebrochene und laufende Instanzen.

Der Rückgabewert der Filter-Funktion ist eine Liste von IData-Objekten. Siehe Auszug 6.2. Aufgebaut ist IData ähnlich einer Key-Value-Map, die ihrerseits beliebig viele IData-Objekte beinhalten kann. Die IData-Liste `instances` ist eine Liste von Instanzen. Jedes

[2] `pub.monitor.process.instance:getInstanceList` in [Sof-2009e, Seite 75]

IData-Objekt wird an die Funktion `tool_listInstance` übergeben und dort auf der Standardausgabe ausgegeben.

Ein Beispiel für diese Ausgabe ist im Auszug 6.3 dargestellt. Angenommen `pm_List` solle alle Instanzen ausgeben, die gestartet sind und bei denen der Kunde „Kunde A“ ist, dann wird in diesem Beispiel die Instanz mit der Instanz-ID `e3f83eb3-4f50-4603-8665-3f8f30218780` als zutreffend erkannt.

```
-----------PROCESS-------------
Instance ID: e3f83eb3-4f50-4603-8665-3f8f30218780
Status     : Started
Model      : EDBPM/ImportWarenverteillager
Inst.Iter. : 1
Image URL  : /WmMonitor/images/processes/process_image12956013750531.
    svg
---------END--PROCESS----------
```

Auszug 6.3: Ausgabe von `pm_List`

Die Ausgabe zeigt die Prozess-ID, den Prozesszustand, das Prozessmodell, die Iteration der Prozessinstanz (vgl. Abschnitt 6.7) und den Pfad eines Prozessdiagrammes mit dem Prozessmodell in der aktuellen Ausführung.

6.4 Start einer neuen Prozessinstanz

Für den Start einer Prozessinstanz wird ein entsprechendes Eingangsdokument publiziert. Die Aufgabe der `pm_Start`-Funktion ist es, das Eingangsdokument zu publizieren und aus den übergebenen Parametern generiert werden.

Parameter	*Wert*
`action`	„Start“
`processtype`	Modellname
`solutions`	Felder des Eingangsdokuments
`conditions`	Keine, da keine laufenden Prozessinstanzen betroffen sind

Tabelle 6.3: Parameter beim Start einer Prozessinstanz

Der Aufbau eines Prozessdokuments ist eine Menge von Schlüsseln (Keys) und Werten (Values). Werden diese Key-Value-Paare als Parameter übergeben, müssen sie zuerst in ein solches Dokument umgewandelt werden. Dazu wird ein XML-Dokument erstellt,

welches als Eingang für die Funktion `tool_submitDocFromXML` dient, die dieses XML-Dokument in ein publizierbares Dokument umwandelt. Das Beispiel in Auszug 6.4 zeigt das Dokument `doc:SendungDokument` in XML-Form.

```
<?xml version="1.0"?>
   <Sendung>5888</Sendung>
   <Sender>Trade Company A</Sender>
   <Herkunft>Hong Kong</Herkunft>
   <Kunde>Unternehmen A</Kunde>
   <Adresse>64283 Darmstadt</Adresse>
   <Ladeliste>...</Ladeliste>
```

Auszug 6.4: Publizierbares XML-Dokument

Die Funktion `tool_submitDocFromXML` erhält den Namen und Inhalt des zu publizierenden Dokuments. Anschließend wird der Inhalt als XML-String mit dem API-Service `xmlStringToXMLNode`[3] in ein XML-Node-Objekt umgewandelt. Dieses XML-Node kann mithilfe des Services `xmlNodeToDocument`[4] wiederum in ein publizierbares Dokument umgewandelt werden. Mit dem API-Service `publish`[5] wird das Dokument publiziert.

Durch den aufgerufenen Service wird das Dokument publiziert. Für jedes publizierte Dokument wird eine Prozessinstanz eines assoziierten Prozessmodells gestartet. Die Betonung liegt hier auf dem Prozessmodell, da es durchaus mehrere Prozessmodelle geben kann, die das publizierte Dokument als ihr Eingangsdokument betrachten. In so einem Fall werden mehrere Prozesse gestartet. Idealerweise ist jeder Dokument-Typ mit nur einem Prozessmodell assoziiert.

6.5 Prozesssteuerung laufender Instanzen

Die Prozesssteuerung laufender Instanzen ist trivial, weil für diesen Fall bereits Services der webMethods-API bereitstehen. Die Funktion `pm_ControlFlow` kapselt die bereitgestellte Funktionalität und implementiert den Start, den Stop und die Wiederaufnahme einer laufenden Prozessinstanz.

[3] `pub.xml:xmlStringToXMLNode`. Der Service ist nicht dokumentiert.
[4] `pub.xml:xmlNodeToDocument` in [Sof-2009d, Seite 686]
[5] `pub.publish.publish` in [Sof-2009d, Seite 385]

Parameter	*Wert*
`action`	„Pause“, „Stop“, „Resume“
`processtype`	Modellname
`solutions`	-
`conditions`	Werte des Daten-Objekts der Instanzen und/oder Instanz-Zustand

Tabelle 6.4: Parameter bei der Prozesssteuerung laufender Instanzen

Die Prozessausführung kennt zahlreiche Zustände für laufende Prozessinstanzen. [Sof-2009e, Seite 196]. Zum Teil sind diese Zustände Markierungen für interne technische Lösungen. Diese Implementierung nutzt nur die drei genannten Zustände, da diese gebräuchlich sind und nur diese benötigt werden.

Die Funktion `pm_ControlFlow` wird vom Prozessmanipulator immer dann aufgerufen, wenn die Aktion gleich „Start“, „Stop“ oder „Resume“ ist. Jede Instanz der Menge betroffener Instanzen wird mitsamt des gewünschten Zustands an die Funktion `tool_changeInstanceStatus` weitergegeben. Die Funktion kapselt den API-Service `chan_geInstanceStatus`[6], der mit der Instanz-ID und der Modell-ID als Parameter die Instanz in den übergebenen Zustand versetzt.

Die betroffenen Instanzen werden ohne Rückmeldung durch den API-Service in den gewünschten Zustand versetzt. Sollte versucht werden eine bereits pausierte Instanz erneut zu pausieren, wird diese Anweisung stillschweigend ignoriert.

6.6 Prozessmodell-Austausch

Die Funktion `pm_Replace` erlaubt den Austausch des Prozessmodells zu jedem Zeitpunkt der Ausführung. Wie in der Ansatz-Beschreibung bereits angemerkt wurde, betrifft der Austausch jedoch das Prozessmodell für alle Instanzen und nicht wie gewünscht nur ausgewählte Prozessinstanzen. Zudem sind Änderungen am Modell nicht temporär, sondern konstant und betreffen auch alle nachfolgenden Instanzen. Um die Veränderungen zur Flexibilität auf bestimmte Prozessinstanzen zu beschränken, muss die Funktion ein Prozessmodell generieren, dass betroffene von unbetroffenen Instanzen unterscheidet. So gesehen unterscheidet sich diese Funktion von den Anderen, da sie nicht nur API-Services aufruft, sondern selbst umfangreiche Funktionen implementiert. Die Funktion arbeitet mit drei Typen von Prozessmodellen: Ein aktuelles (altes) Prozessmodell, ein verändertes (neues) Prozessmodell und ein flexibilisiertes (generiertes) Prozessmodell als Delta der Vorgänger.

[6] `pub.monitor.process.instanceControl:changeInstanceStatus` in [Sof-2009e, Seite 110]

Parameter	*Wert*
`action`	„Replace Model“
`processtype`	Modellname
`solutions`	Das aktuelle und veränderte Prozessmodell als XML-Modelle
`conditions`	Werte des Daten-Objekts der Instanzen und/oder Instanz-Zustand

Tabelle 6.5: Parameter beim Austausch des Prozessmodells

Die Funktion benötigt als Parameter das aktuelle Prozessmodell der zu flexibilisierenden Instanz und das veränderte Modell. Aus diesem Modell wird mithilfe der ermittelten Prozess-IDs ein flexibilisiertes Modell generiert. Das flexibilisierte Prozessmodell ist eine Kombination aus dem alten und veränderten Modell. Es verfügt über die sämtliche Prozesselemente beider Modelle, auch der Elemente, die verändert wurden. Die Kombination der Modelle wird um Sequenzflüsse und Sequenzflussbedingungen erweitert, die das Modell weiterhin gültig und ausführbar machen.

Zunächst soll der XML-Aufbau des Prozessmodells beschrieben werden. Anhand der XML-Repräsentation des Modells wird das Delta zwischen den beiden überlieferten Modellen ermittelt. Mithilfe dieses Deltas wird dann das flexibilisierte XML-Modell generiert.

6.6.1 XML-Darstellung des Prozessmodells

In webMethods wird ein Prozessmodell in einem proprietären XML-Format dargestellt. Die Tabelle 6.6 zeigt den Zusammenhang zwischen dem BPMN-Standard und der Prozess-Repräsentation, wie sie in webMethods implementiert ist. Aus der Tabelle geht hervor, dass immer mehrere Prozesselemente sich ein XML-Element teilen, mithilfe dessen sie dargestellt werden. Das führt dazu, dass beispielsweise auf `invokeSteps` transparent zugegriffen werden kann. Unabhängig davon, ob es sich dabei um „Tasks“ oder „User Tasks“ handelt.

Das webMethods-Prozessmodell besteht aus dem Root-Knoten `businessProcessDia_gram` und untergeordneten Kind-Elementen vom Typ `pool`. Hinzu kommen auszugsweise die Kind-Elemente `systemKPIs`, `timeout`, `documentation`, `qualitiyOfService`, die Prozess-Eigenschaften und -Bedingungen darstellen. Im Pool-Element sind die Prozesselemente abgelegt. Alle Elemente verfügen über einen eindeutigen Bezeichner und werden zudem über ein kartesisches Koordinatensystem im Diagramm platziert. Damit ist das Prozessmodell nicht nur Modell, sondern auch Prozess-Diagramm zugleich. Innerhalb des Pool-Elements gibt es eine Menge von Step-Elementen, dazu zählen `receiveStep`, `terminateStep` und `invokeStep`. Als Start des Prozesses wird im `receiveStep` das

BPMN 2.0		*webMethods Process*	
Modell	*Ausprägung*	*XML-Element*	*Implementierung*
Aktivität	Task	invokeStep	Task
	Manueller Task		Manual Task
	Benutzer Task		User Task
	Service-Task		Service Task
	Geschäftsregel-Task		Business Rule Task
	Skript-Task		-
	Empfangs-Task		Receive Task
	Sende-Task		Send Task
	Globaler Task		-
	Unterprozess		Call Activity
	Teilprozess	container	Subprocess
	Ereignisbasierter Teilprozess	-	-
	Ad-hoc-Teilprozess	-	-
	Transaktion	-	-
Nachrichtenfluss	Sequenzfluss	transition	Pfad
	Bedingter Fluss		IF-Pfad
	Standardfluss		ELSE-Pfad
	Nachrichten-Fluss	-	-
Ereignisse	Start-Ereignis	invokeStep	Message-Event
	End-Ereignis	terminateStep	Terminate
	Zwischen-Ereignis	-	
Gateway	parallel (XOR)	gatewayStep	XOR-Join
	komplex (UND/ODER)		AND/OR-Join
	ereignisbasiert	-	-
	ereignisbasiert (XOR)	-	-
Daten	Daten-Objekt	-	Process Pipeline
	Dateneingabe	-	-
	Datenausgabe	-	-
	Datenspeicher-Referenz	-	-
	Datenassoziation	-	-

Tabelle 6.6: Zusammenhang zwischen der BPMN und webMethods-Prozessmodellen

Daten-Objekt spezifiziert. Die `invokeSteps` und `gatewaySteps` stellen Aktivitäten und Gateways dar und werden durch eigene Kind-Elemente spezifiziert.

Nach den verschiedenen Steps folgen die Transitions, die Sequenzflüsse darstellen. Transitions verfügen neben ihrem eindeutigen Bezeichner über die Attribute `source` und `target`. Diese Attribute referenzieren die eindeutigen Bezeichner der Quell- und Ziel-Step-Elemente. Transitions können Bedingungen beinhalten, die sie zu bedingten Sequenzflüssen machen. Diese Bedingungen sind Kind-Elemente der Transitions und werden logisch miteinander verknüpft.

6.6.2 Modell-Instanz-Problematik

Wie bereits in der Ansatzbeschreibung ermittelt wurde, sollen einzelne Instanzen mithilfe des Prozessmodells flexibilisiert werden. Dazu müssen Meta-Sequenzflüsse in das allgemeine Prozessmodell eingebaut werden, die für betroffen und unbetroffene Instanzen ein unterschiedliches Verhalten ermöglichen. Trotz der Flexibilisierungen soll das Prozessmodell ausführbar und gültig bleiben.

6.6.3 Flexibilisierung der Prozesselemente

Zur Flexibilisierung der einzelnen Prozesselemente müssen zuerst die Unterschiede im Basis-Modell und im veränderten Modell erkannt werden. Dazu werden die Prozesselemente der beiden Modelle miteinander verglichen und die unterschiedlichen Elemente in die Menge der veränderten Elemente aufgenommen.

In der Implementierung wird dazu zuerst die Menge der unveränderten Elemente mithilfe der Funktion `flex_getUnchangedElements` bestimmt. Die Funktion vergleicht sämtliche Elemente beider Prozessmodelle aufgrund ihrer Elemente und Attribute miteinander. Ausnahmen sind dabei die Koordinaten des Elements und die Koordinaten des Ziel-Elements bei Sequenzflüssen. Die Modell-Koordinaten sind fragil und können schon bei minimalen Änderungen am Modell auch bei unbetroffenen Elementen verändert werden. Da die grafische Darstellung des Prozessmodells nicht die Ausführung des Modells beeinflusst, werden die veränderten Koordinaten ignoriert.

Ist ein Prozesselement aus dem veränderten Modell identisch mit dem Element aus dem aktuellen Modell, wird es in die Menge der unveränderten Elemente eingereiht. Von den unveränderten Elementen wird anschließend auf die veränderten Elemente geschlossen. Die Elemente des alten und aktuellen Prozessmodells, die nicht in der unveränderten Menge sind, müssen verändert worden sein.

Im nächsten Schritt muss bestimmt werden, ob ein verändertes Element gelöscht, hinzugefügt oder bearbeitet wurde. Der Schritt ist wichtig, da je nach Fall unterschiedlich flexibilisiert wird. Fehlt ein Element des *alten* Modells in der Menge der veränderten Elemente, ist es gelöscht worden. Fehlt ein Element des *veränderten* Modells in der Menge der alten Elemente, ist es hinzugefügt worden. Aufwendiger ist es, bearbeitete Elemente zu erkennen. Ist ein verändertes Element im alten und im veränderten Modell enthalten, muss geprüft werden, ob es bearbeitet wurde. Hierzu werden sämtliche Attribute und Kind-Elemente des Elements miteinander verglichen. Unterscheiden sich Attribute oder Kind-Elemente eines Elements, ist es bearbeitet worden.

An diesem Punkt wird ein flexibilisiertes Prozessmodell generiert, das zu Anfang mit dem alten Prozessmodell identisch ist. Die folgenden Änderungsarten erweitern auf unterschiedliche Weise das flexibilisierte Modell um Elemente und Sequenzflüsse aus dem veränderten Prozessmodell. Das flexibilisierte Prozessmodell wird immer nur erweitert, da Änderungen für unflexibilisierte Prozessinstanzen „unsichtbar“ sein sollen. Prozesselemente werden auf XML-Ebene vom veränderten in das flexibilisierte Modell kopiert.

Gelöschte Elemente Diese Änderung ist die einfachste der drei Änderungsfälle. Das gelöschte Element wird aus dem Basis-Modell in das flexibilisierte Modell kopiert und für flexibilisierte Instanzen unzugänglich gemacht. Für zum Element hinführende Sequenzflüsse heißt das, dass sie mit einer logischen Ungleichheit versehen werden. Also nur nicht-flexibilisierte Instanzen auf das neuerdings gelöschte Element zugreifen können. Sequenzflüsse, die vom gelöschten Element wegführen, bekommen keine Einschränkung. So wird verhindert, dass Prozessinstanzen auf einem gelöschten Element ohne Ausweg aktiv sind. In diesem Fall könnten sie nicht mehr weiter am Prozessablauf teilhaben und wären „gefangen“.

Hinzugefügte Elemente Das neue Element wird inklusive hin- und wegführender Sequenzflüsse aus dem veränderten Modell in das flexibilisierte Modell kopiert. Die Sequenzflüsse, die zum Element hinführen und ebenso aus dem veränderten Modell stammen, werden für alle unflexibilisierten Instanzen unzugänglich gemacht. Damit können nur flexibilisierte Instanzen zum hinzugefügten Element gelangen. Wegführende Sequenzflüsse bleiben bedingungslos.

Bearbeitete Elemente Ist ein Element in der Menge der veränderten Elemente und tritt es in beiden Modellen auf, ist es in seinen Eigenschaften bearbeitet worden. Um das unbearbeitete Element für alle unflexibilisierten Instanzen zu behalten, wird dieses aus dem *unveränderten* Modell in das flexibilisierte Modell kopiert. Zudem wird aus dem

veränderten Modell die bearbeitete Version des Elements in das flexibilisierte Modell kopiert.

Das Problem dieser Element-Duplizierung ist, das zwei Elemente mit demselben eindeutigen Element-Bezeichner im Prozessmodell existieren. Um das Modell weiterhin konsistent zu halten, wird das veränderte Modell mit einem neuen eindeutigen Bezeichner angelegt. Dazu wird dem Bezeichner das Suffix „flex“ angefügt. Im Anschluss werden sämtliche Sequenzflüsse zum und vom ursprünglichen Element dupliziert und deren ID ebenfalls um das Suffix erweitert. Damit diese zusätzlichen Sequenzflüsse auf das unveränderte Element zeigen, werden die `Source`- und `Target`-Attribute des Elements entsprechend angepasst. Das unveränderte Element bleibt nur für unflexibilisierte Instanzen zugänglich und das veränderte Element nur für flexibilisierte Instanzen.

Sequenzfluss-Flexibilisierung

Die Sequenzfluss-Flexibilisierung teilt sich in in die Behandlung von Meta-Sequenzflüssen und Sequenzflüssen des Prozessmodells.

Die Funktion `flex_addConditions` fügt Sequenzflüssen auf Meta-Ebene, je nachdem ob sie betroffen sind oder nicht, „!=„ oder „==„ und eine Liste der betroffenen Instanzen die entsprechende Bedingung hinzu. Auf diese Weise können Sequenzflüsse für Prozessinstanzen zugänglich oder unzugänglich gemacht werden. Durch die Funktion `flex_addConditions` werden zudem die logischen Probleme gelöst, die im Zusammenhang mit der funktionalen Ebene und der Meta-Ebene der Flexibilisierung entstehen.

Neben Prozesselementen können zudem Sequenzflüsse des Prozessmodells verändert worden sein. Damit nicht bereits flexibilisierte Sequenzflüsse fälschlich als verändert erkannt werden, müssen die veränderten Sequenzflüsse noch vor der Element-Flexibilisierung und der Veränderung der Sequenzflüsse auf der Meta-Ebene abgefragt werden. Die Abfrage funktioniert auf die gleiche Weise wie die Abfrage der veränderten Prozesselemente. Die Flexibilisierung von Sequenzflüssen ist einfacher als die der Prozesselemente, da Sequenzflüsse selbst keine an- und weggehenden Sequenzflüsse haben und so nicht von anderen Elementen abhängig sind.

Hinzugefügte Sequenzflüsse werden nur für betroffene Instanzen zugänglich gemacht. Gelöschte Sequenzflüsse dagegen nur für unbetroffene Instanzen. Ist ein Sequenzfluss bearbeitet worden, wird er dupliziert und die bearbeitete Version für betroffene Instanzen zugänglich gemacht. Dementsprechend wird die unbearbeitete Version nur für unbetroffene Instanzen zugänglich gemacht. Zudem wird die ID des flexibilisierten Sequenzflusses um die Suffix „flex“ erweitert.

6.6.4 Aufspielen des flexibilisierten Prozessmodells

Nach diesen Schritten liegt ein flexibilisiertes Prozessmodell vor. Da das flexibilisierte Prozessmodell weitestgehend auf dem alten Prozessmodell besteht und das Modell programmatisch konsistent gehalten wurde, kann es aufgespielt werden.

Dazu wird die Funktion `tool_deployProcessModel` verwendet. Die Funktion kapselt den API-Service `generateProcess`[7] so, dass ein Prozessmodell als Parameter übergeben werden kann. Zudem kann die Version des Prozessmodells als Parameter übermittelt werden. Der API-Service geht davon aus, dass ein Prozessmodell als Datei auf der Festplatte der Prozessausführungsumgebung vorliegt. `tool_replaceProcessModel` nimmt das übergebene flexibilisierte Prozessmodell und schreibt es an einen temporären Pfad, der dem API-Service bekannt ist, und löst ihn aus. Der API-Service liest das Modell aus und spielt es über diesen Umweg auf.

Nachdem das flexibilisierte Prozessmodell mit der Funktion `tool_deployProcessModel` in der bestehenden Version aufgespielt wurde, wird der Versions-Parameter des Modells inkrementiert und das alte Prozessmodell ebenso aufgespielt. Dieser Schritt sorgt dafür, dass zukünftige Prozessinstanzen wieder auf dem unflexibilisierten Prozessmodell ablaufen. Optional kann der Schritt ausgelassen werden, dann sind auch alle zukünftigen Instanzen flexibel.

Damit laufende Prozessinstanzen in das flexibilisierte Prozessmodell migriert werden können, wird die neue Version *deaktiviert* aufgespielt. Sobald

Die Migration von laufenden Instanzen funktioniert jedoch nicht in der beschriebenen Weise. Siehe Abschnitt 6.9.

6.7 Wiederholung von Aktivitäten

Zur Wiederholung von Aktivitäten und zum Hinzufügen einer zusätzlichen Aktivitäts-Instanz werden die Funktionen `pm_Redo` und `pm_NewActivityInstance` verwendet. Die beiden Funktionen sind identisch und unterscheiden sich nur dadurch, dass bei der Funktion `pm_Redo` vor dem Resubmit eines Steps die laufende Prozessinstanz abgebrochen wird.

[7] `wm.designer.ProcessGenerator:generateProcess`. Der Service ist nicht dokumentiert.

Parameter	*Wert*
`action`	„Redo“ / „New Activity Instance“
`processtype`	Modellname
`solutions`	Name der zu wiederholenden Aktivität
`conditions`	Werte des Daten-Objekts der Instanzen und/oder Instanz-Zustand

Tabelle 6.7: Parameter bei der Wiederholung von Aktivitäten

Zuerst soll die Funktion `pm_NewActivityInstance` beschrieben werden, die auch die Basis für `pm_Redo` bildet. Nachdem die Menge der betroffenen Prozessinstanzen ermittelt wurde, wird für jede ermittelte Instanz das folgende Verhalten angewandt. Zuerst muss der übergebene Name der Aktivität umgewandelt werden. Durch die API werden Aktivitäten nämlich nicht anhand ihrer Namen adressiert, sondern aufgrund einer internen Step-ID. Um die Step-ID für einen Aktivitäts-Namen zu ermitteln, werde alle Aktivitäten des betroffenen Prozessmodells über die Funktion `tool_getSteps` ermittelt und deren Namen mit dem übergebenen Aktivitätsnamen verglichen. Ist die gewünschte Aktivität gefunden, wird mit ihrer Step-ID und der Instanz-ID die Funktion `tool_resubmit` gestartet. `tool_resubmit` kapselt den API-Service `resubmitInstanceStep`[8]. Nach der Ausführung wird die gewählte Instanz an der Stelle der Aktivität durch einen zusätzlichen Sequenzfluss neu eingesetzt.

Bei der Funktion `pm_Redo` wird zusätzlich mit der bereits bekannten Funktion `tool_changeInstanceStatus` die laufende Prozessinstanz gestoppt. Anschließend wird die gestoppte Instanz durch `tool_resubmit` neu eingesetzt. Dieser Schritt kommt einer Wiederholung der gewünschten Aktivität gleich.

Die `tool_resubmit`-Funktion verfügt zudem über einen Pipeline-Parameter, über den ein verändertes Daten-Objekt übergeben werden kann. In den beiden obigen Fällen wurde ein leeres Pipeline-Objekt übergeben, damit nichts verändert wird. Soll das Daten-Objekt hingegen verändert werden, muss der Pipeline-Parameter definiert sein. Bei der folgenden Funktion `pm_Update` ist das der Fall.

6.8 Änderungen am Datenobjekt

Die Funktion `pm_Update` ändert das Daten-Objekt von betroffenen Instanzen und ermöglicht damit den Ansatz zum Ändern des Daten-Objekts und den Ansatz für dynamische Service-Referenzen.

[8] `pub.monitor.process.instanceControl:resubmitInstanceStep` in [Sof-2009e, Seite 113]

Parameter	*Wert*
`action`	„Update“
`processtype`	Modellname
`solutions`	Veränderte Daten-Objekt-Felder mit neuen Werten.
`conditions`	Werte des Daten-Objekts der Instanzen und/oder Instanz-Zustand

Tabelle 6.8: Parameter bei der Wiederholung von Aktivitäten

Für alle betroffenen Instanzen soll das Daten-Objekt angepasst werden. In diesem Fall ist das Solution-Feld mit den veränderten Daten-Objekt-Feldern definiert und optional der zu wiederholenden Aktivität.

Zunächst wird das veränderte Daten-Objekt mithilfe der bereits erwähnten Funktionen `xmlStringToXMLNode` und `xmlNodeToDocument` in ein publizierbares Dokument umgewandelt. Das Dokument wird anschließend als Pipeline-Parameter für die Funktion `tool_resubmit` übergeben, statt wie bei `pm_Redo` ohne den Pipeline-Parameter zu starten. Der Aufbau des Pipeline-Parameters ist unklar und schlecht dokumentiert ([Sof-2009e, Seite 113]). Im folgenden Abschnitt ist dieses Problem genauer beschrieben.

6.9 Einschränkungen

Nach bisherigem Stand (Februar 2011) funktionieren die Komponenten „Prozessmodell-Austausch“ und „Änderung am Daten-Objekt“ des Prototyps nicht oder nicht richtig.

In der aktuellen Version von webMethods bereitet das Aufspielen eines Prozessmodells mehr Schwierigkeiten als angenommen. Der API-Service `generateProcess` bewirkt einen Fehler, der dafür sorgt, dass der Integration Server völlig ausgelastet wird. In der Folge ist es unmöglich, weiter auf dem Server zu arbeiten. Dieses Problem lässt sich dadurch lösen, dass das flexibilisierte Prozessmodell an eine Stelle geschrieben wird, an der Software AG Designer zugreifen kann. Das flexibilisierte Modell im Designer in gewohnter Weise geöffnet und aufgespielt. Nachdem die neue Version des Modells aufgespielt wurde, sollte es laut Dokumentation möglich sein, laufende Instanzen auf die neue Modellversion „upzugraden“. [Sof-2009f][Seite 101] Dieser Schritt misslingt und führt dazu, dass das Prozessmodell nicht mehr ausgeführt werden kann. Intensive Bemühungen und andauernde Versuche haben zu keinem Ergebnis geführt. Stattdessen wurden Fehlermeldungen geworfen, die auf einen Fehler in der Implementierung von webMethods hindeuten.

Zudem funktioniert die Änderung am Daten-Objekt aufgrund eines unklaren Parameters des API-Services `resubmitInstanceStep` nur über Umwege. Obwohl die Pipeline einer

Prozessinstanz über die Prozessverwaltung von „My webMethods“ verändert werden kann, war der Schritt programmatisch nicht möglich. Statt die Pipeline zu verändern, wird der Prozess lediglich mit den unveränderten Werten wiederholt. Die Versuche mit „My webMethods“ ergaben jedoch, dass Daten einer Prozessinstanz zur Laufzeit verändert werden können. Am Beispiel einer dynamischen Service-Referenz konnte zudem nachgewiesen werden, dass nach der Änderung auch dynamische Services möglich sind. Außerdem bietet die Funktion noch keine Lösung für die Frage, an welcher Stelle, des Prozesses eine Prozessinstanz sich befindet. Daher muss zusätzlich zu den Werten der Änderung die zu wiederholende Aktivität übergeben werden.

Ob die Probleme auch in neueren oder zukünftigen webMethods-Versionen auftreten, ist nicht bekannt. Ein solcher Test währe aufgrund der Komplexität der Suite und der umfangreichen Konfigurationen, die für den Prototyp notwendig waren, im Zeitraum der Arbeit nicht zu bewältigen gewesen.

Das folgende Kapitel beschreibt die Integration des Prozessmanipulators zwischen der CEP-Engine und der Prozessausführungsumgebung und geht davon aus, dass die beiden Probleme gelöst werden konnten.

Kapitel 7

Integration des Prototyps

Der entwickelte Prototyp für den Prozessmanipulator stellt die Funktionen bereit, die dazu notwendig sind, Prozessinstanzen zu flexibilisieren. Nun soll der entstandene Prototyp an die Ereignis-Quelle angebunden werden. Hierzu wird im Abschnitt 7.1 das komplexe Ereignis „Prozessmanipulation“ beschrieben, das den den Auslöser des Prozessmanipulators darstellt und seine Eingangsdaten übergibt. Im Abschnitt 7.2 wird gezeigt, wie ein komplexes Ereignis aus einer Menge von Ereignissen aggregiert werden kann. Danach wird in Abschnitt 7.3 die Schnittstelle des Prozessmanipulators besprochen, um komplexe Ereignisse zu empfangen. Zum besseren Verständnis zeigt Abschnitt 7.4 ein vollständiges Beispiel von der Aggregation des komplexen Ereignisses „Prozessmanipulation“ bis hin zur Flexibilisierung einzelner Prozessinstanzen.

7.1 Komplexes Ereignis „Prozessmanipulation“

Das komplexe Ereignis „Prozessmanipulation“ transportiert für den Prozessmanipulator Anweisungen, das betroffene Prozessmodell, Lösungen und Bedingungen. Das komplexe Ereignis wird zuvor in webMethods Business Events modelliert. Der Aufbau eines komplexen Ereignisses ist durch das XML-Schema[1] in Auszug 7.1 definiert. Das komplexe Ereignis verfügt über die Elemente `Action`, `ProcessType`, `Solution` und `Condition`. Über diese Elemente kann der Prozessverantwortliche in die Prozessausführung eingreifen. Das Element `Action` beschreibt eine vordefinierte Funktion des Prozessmanipulators, die ausgeführt werden soll. An dieser Stelle können die im vorherigen Kapitel implementierten Funktionen ausgewählt werden. Das Element `Solution` ist kontextabhängig vom Feld `Action`. Die hier spezifizierte Lösung übermittelt die Funktionsparameter, die die auszuführende Funktion benötigt.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:eda="http://namespaces.softwareag.com/EDA/Event">
  <import namespace="http://namespaces.softwareag.com/EDA/Event"
 schemaLocation="EventEnvelope.xsd"/>
  <element name="ProcessManipulationEvent" substitutionGroup="eda:
      Payload">
    <complexType>
      <sequence>
        <element name="Action" type="string"/>
        <element name="ProcessType" type="string"/>
        <element name="Solution" type="string"/>
          <complexType>
            <sequence>
              <element element name="Item" type="xsd:string"
                       minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
          </complexType>
        <element name="Condition" type="string"/>
          <complexType>
             <sequence>
               <element element name="Item" type="xsd:string"
                        minOccurs="0" maxOccurs="unbounded"/>
             </sequence>
          </complexType>
      </sequence>
    </complexType>
  </element>
</schema>
```

Auszug 7.1: XML-Schema für komplexes Ereignis „Prozessmanipulation“

[1] Ein XML-Schema modelliert die Struktur eines XML-Dokuments.

Die Tabelle 7.1 zeigt die Beziehung zwischen Action und Solution nach Abhängigkeit der gewählten Action. Wenn beispielsweise eine Update-Action ausgewählt wird, müssen die neuen Variablen im Solution Feld übergeben werden. Im Beispiel für den Start wird eine Prozessinstanz gestartet bei der der Sender gleich „Trade Company A“ ist und der Kunde gleich „Unternehmen A“ ist usw.

Action	*Solution*	*Beispiel*
List	-	-
Start	Variable 1 Variable 2 Variable 3 Variable 4 ...	„<Item id=„Sender">Company A</Item>“ „<Item id=„Kunde">Kunde A</Item>“ „<Item id=„Adresse">64283 Darmstadt</Item>“ „<Item id=„Herkunft">Hong Kong</Item>“ ...
Stop	-	-
Pause	-	-
Resume	-	-
Replace Model	Altes Modell Neues Modell	<Item id=„oldmodel"><bpmn>...</bpmn></Item> <Item id=„newmodel"><bpmn>...</bpmn></Item>
New Act. Inst.	Name	„<Item>Transport-Auftrag zum Kunden</Item>“
Redo	Name	„<Item>Transport-Auftrag zum Kunden</Item>“
Update	Variable 1 Variable 2 Variable 3 Variable 4 ...	„<Item id=„Sender">Trade Company B</Item>“ „<Item id=„Kunde">Unternehmen B</Item>“ „<Item id=„Adresse">60308 Frankfurt</Item>“ „<Item id=„Herkunft">Singapur</Item>“ ...
Update	Referenz	„<Item id=„Service1>extTransport</Item>“

Tabelle 7.1: Beispiele für Solutions des Prozessmanipulations-Ereignisses

Action	*Condition*	*Beispiel*
List	Zustand Variable 1 Variable 2 Variable 3 ...	„<Item>Started</Status>“ „<Item id=„Kunde">Unternehmen A</Item>“ „<Item id=„Adresse">64283 Darmstadt</Item>“ „<Item id=„Herkunft">Hong Kong</Item>“ ...
Start	”	”
Stop	”	”
Pause	”	”
Resume	”	”
Replace Model	”	”
New Act. Instance	”	”
Redo	”	”
Update	”	”

Tabelle 7.2: Beispiele für Conditions des Prozessmanipulations-Ereignisses

In Tabelle 7.2 am Beispiel der Action List optionale Werte für das Condition-Feld gezeigt. Die Bedingungen sind konjugiert. Analog können für die anderen Actions die gleichen Conditions definiert werden. Kombiniert man die Werte der beiden Tabellen, ergibt sich z. B. bei „Redo“ die folgende Anweisung und das komplexe Ereignis in Auszug 7.2:

Wiederhole die Aktivität mit dem Namen „Transport-Auftrag zum Kunden“ für alle Prozessinstanzen die gestartet wurden und deren Kunde „Unternehmen A“, deren Adresse „64283 Darmstadt“ und deren Herkunft „Hong Kong“ ist usw.

```
<ProcessManipulationEvent>
    <Action>Redo</Action>
    <ProcessType>EDBPM/Import-Warenverteillager</ProcessType>
    <Solution>Transport-Auftrag zum Kunden</Solution>
    <Condition>
        <Item>Started</Item>
        <Item id="Kunde">Unternehmen A</Item>
        <Item id="Adresse">64283 Darmstadt</Item>
        <Item id="Herkunft">Hong Kong</Item>
        <...>...</...>
</ProcessManipulationEvent>
```

Auszug 7.2: Anweisung für Redo

Im Anhang dieser Arbeit auf Seite 122 sind weitere beispielhafte Prozessmanipulations-Ereignisse aufgeführt, die Variationen von Actions, Solutions und Conditions enthalten.

7.2 Aggregation des komplexen Ereignisses

Die *Event Query Language* (EQL) ist die Event-Processing Language von webMethods Business Events. Die EQL dient dazu, Ereignisse zu aggregieren und komplexe Ereignisse zu erstellen. Zudem werden durch die Query die Felder `Action`, `ProcessType`, `Soluti_on` und `Condition` des Ereignisses „Prozessmanipulation“ gefüllt. Die EQL ist in einer Sprache formuliert, die an SQL angelehnt ist. [Kr2007] Die EQL wird hier nur anhand einiger Beispiele vorgestellt, da sie nicht zentraler Bestandteil der Arbeit ist.

In der Ereignis-Query können die bekannten SQL-Statements `SELECT`, `FROM` und `WHERE` auftreten. Im `SELECT`-Statement werden die auszugebenden Werte des komplexen Ereignisses als Ergebnis spezifziert. Anhand des `FROM`-Statements wird die Menge der zu betrachtenden Basis-Ereignisse definiert. Das `WHERE`-Statement definiert Bedingungen für zu betrachtende Basis-Ereignisse. Der Auszug 7.3 zeigt Beispiel-Querys.

In der ersten Beispiel-Query wird die längste Verspätung der letzten fünf Verspätungen ermittelt. Das daraus resultierende komplexe Ereignis enthält die maximale Dauer einer

```
--- Query 1
SELECT MAX(Dauer) FROM VERSPAETUNGEN WINDOW(RANGE 5);

--- Query 2 KritischerKunde
SELECT KundeID
FROM (SELECT COUNT(*) AS Anzahl, KundeID
      FROM Verspaetung WINDOW(Range 3 Weeks) GROUP BY KundeID) AS
          SUBQUERY
WHERE Anzahl > 3;

--- Query 3
SELECT 'Start',
'EDBPM/ImportWarenverteillager',
'',
'<Item id="Sender">'|| Sendung.Sender ||'</Item>
<Item id="Kunde">'|| Sendung.Kunde ||'</Item>
<Item id="SendungsID">'|| Sendung.SendungsID ||'</Item>'
FROM Sendung;
```

Auszug 7.3: Ereignis-Query in der WM-EPL

Verspätung. Das zweite Beispiel zeigt die Query für das Ereignis „Kritischer Kunde“ die mit einer Sub-Query realisiert ist. Übersteigt die Anzahl der Verspätungen in den letzten drei Wochen pro Kunde die Anzahl drei, wird das komplexe Ereignis mit der Kunden-ID generiert.

Das dritte Beispiel zeigt schließlich ein komplexes Ereignis des Typs „Prozessmanipulation“. Die Felder `Action`, `ProcessType`, `Solutions` und `Conditions` werden innerhalb des `SELECT`-Statements definiert. Zudem wurde die Query um Referenzen auf das Ereignis „Sendung“ erweitertet, die zum Start in das Solutions-Feld eingefügt werden. Anzumerken ist hier, dass die Werte von Solutions und Conditions als verkettete XML-Elemente in String-Form angegeben werden. Das XML-Schema für komplexe Ereignisse erlaubt zu diesem Zeitpunkt keine Sub-Elemente von Ereignis-Elementen.

7.3 Schnittstelle zum Prozessmanipulator

Nachdem die Query zur Aggregation von komplexen Ereignissen und den Aufbau eines komplexen Ereignisses beschrieben wurde, wird in diesem Abschnitt beschrieben, wie die komplexen Ereignisse von der CEP-Engine zum Prozessmanipulator gelangen.

Damit ein Ereignis-Query aktiviert werden kann, benötigt es neben der eigentlichen Query weitere Einstellungen. Der Auszug 7.4 zeigt den Inhalt einer vollständigen und lauffähigen Query. Nach dem Query-`Name` wird im Feld `EventType` der Typ des komplexen

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Query>
    <Name>KritischerKunde</Name>
    <EventType>KritischerKunde</EventType>
    <QueryString>SELECT KundeID &#xD;
FROM (SELECT COUNT(*) AS Anzahl, KundeID&#xD;
            FROM Verspaetung WINDOW(Range 3 Weeks) GROUP BY KundeID)
                AS SUBQUERY&#xD;
WHERE Anzahl &gt; 3;    &#xD;
    </QueryString>
    <ConsoleOutput>false</ConsoleOutput>
    <JMSOutput>true</JMSOutput>
    <Binding>
        <Broker>Broker #1</Broker>
        <Destination>Event::ProcessManipulator</Destination>
        <DestinationType>TOPIC</DestinationType>
    </Binding>
</Query>
```

Auszug 7.4: Vollständiger Aufbau einer Ereignis-Query

Ereignisses angegeben. Der Typ entspricht dem komplexen Ereignis `ProcessManipula_tionEvent` , das in der XML-Schema-Definition modelliert wurde. Das `QueryString`-Element beinhaltet die Query, wie sie im vorherigen Abschnitt vorgestellt wurde.

Durch die Felder `ConsoleOutput` und `JMSOutput` wird die Art der Ausgabe des komplexen Ereignisses definiert. Da das komplexe Ereignis über den Java Message Service übertragen werden soll, wird der entsprechende Output aktiviert. webMethods Business Events empfängt und sendet Ereignisse über einen Topic des Java Message Service (JMS). Tritt ein Ereignis am Topic auf, wird der Prozessmanipulator gestartet. Der Prozessmanipulator-Service interpretiert den Inhalt des Ereignisses und flexibilisiert betroffene Prozesse entsprechend der dort enthaltenen Anweisungen. Tatsächlich wird der Service nicht nur bei komplexen Ereignissen des Typs „Prozessmanipulation“ gestartet, sondern bei allen Ereignissen auf dem Topic. Die CEP-Engine muss komplexe Ereignisse also so senden, dass nur Ereignisse vom Typ „Prozessmanipulation“ der spezielle Topic verwendet wird. Sollte der Prozessmanipulator durch ein anderes Ereignis als einer Prozessmanipulation gestartet werden, kann er alternativ abbrechen. Die Kind-Elemente des Feldes `Binding` definieren die Verbindungsdaten zum webMethods Broker, der den Topic `Event::ProcessManipulator` bereitstellt.

7.4 Vollständiges Beispiel

Nachdem eine Sendung aus dem gerade angekommenen Flugzeug ausgeladen wurde, wird mithilfe der Query 3 aus dem Auszug 7.3 das komplexe Ereignis in Auszug 7.5 generiert. Sobald das komplexe Ereignis beim Prozessmanipulator ankommt, wird eine Prozessinstanz mit den Werten aus dem Condition-Feld generiert. Die Instanz bekommt von der Prozessausführung die ID „a0“ zugewiesen. Dieser Schritt wird für alle Sendungen des Fluges ausgeführt.

```
<ProcessManipulationEvent>
    <Action>Start</Action>
    <ProcessType>EDBPM/Import-Warenverteillager</ProcessType>
    <Solution></Solution>
    <Condition>
        <Item id="Sender">Transport Company A</Item>
        <Item id="Kunde">Kunde A</Item>
        <Item id="SendungsID">4711</Item>
        <Item id="Flug">5888</Item>
</ProcessManipulationEvent>
```

Auszug 7.5: Komplexes Ereignis für den Start einer Prozessinstanz

Das vollständige Beispiel zeigt den Ablauf einer Prozessflexibilisierung aus dem motivierenden Beispiel von Seite 3 verwendet. Zur Erinnerung: Die verspäteten Sendungen eines kritischen Kunden, der in den vergangenen Tagen bereits von Verspätungen betroffen war, sollen direkt vom Zoll aus zum Kunden gesendet werden.

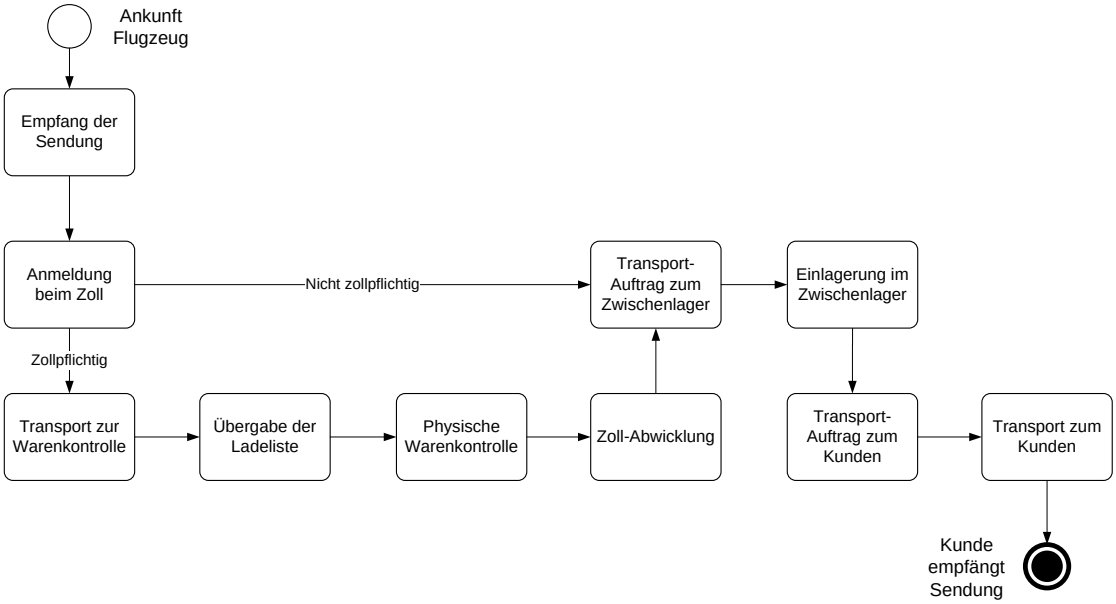

Abbildung 7.1: Unveränderter Prozess „Import-Warenverteillager“

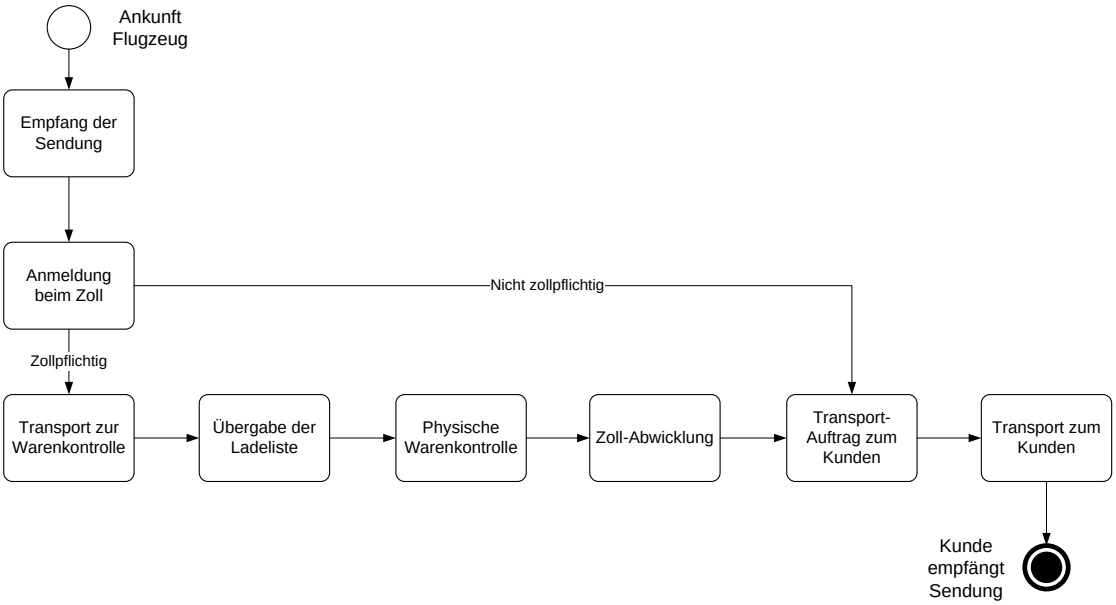

Abbildung 7.2: Veränderter Prozess „Import-Warenverteillager“

7.4.1 Ereignis-Query

Um aus den Teilereignissen „Flug verspätet“ und „Kunde war bereits von Verspätungen betroffen“ das komplexe Ereignis „Prozessmanipulation“ zu generieren, muss eine Ereignis-Query definiert werden. Die Ereignis-Query definiert sowohl die Bedingungen für den Eintritt des komplexen Ereignisses als auch den Inhalt des Ereignisses selbst. Der folgende Auszug zeigt den Aufbau dieser Query.

```
SELECT 'Replace Model',
       'EDBPM/ImportWarenverteillager',
       '<Item id="oldmodel"><![CDATA[...]]></Item>
        <Item id="newmodel"><![CDATA[...]]></Item>',
       '<Item id="Kunde">'|| Verspaetung.KundeID ||'</Item>'
FROM Verspaetung, KritischerKunde WINDOW(3 Weeks)
WHERE Verspaetung.KundenID = KritischerKunde.KundenID
```

Auszug 7.6: CEP-Query für das Ereignis „Prozessmanipulation“

Im `FROM`-Feld wird die Menge der zu berücksichtigenden Ereignisse definiert. Das `WHERE`-Feld stellt die Bedingung für den Eintritt des komplexen Ereignisses. Im vorliegenden Fall ist dies, ob eine Flugverspätung vorliegt und ob ein betroffener Kunde in den letzten drei Wochen bereits von verspäteten Sendungen betroffen war. Ist dies der Fall, wird ein komplexes Ereignis mit den durch das `SELECT`-Statement definierten Feldern erstellt. Zur Lösung des Problems sollen alle Instanzen mit dem kritischen Kunden flexibilisiert werden. Die Flexibilisierung ist durch das Feld `newmodel` definiert, das ein verändertes Prozessmodell zur Lösung des Problems beinhaltet. Damit die nötigen Parameter für die Action „Replace Model“ bereitstehen, wird mit `oldmodel` auch das aktuelle Prozessmodell übertragen. Das daraus entstehende komplexe Ereignis sieht folgendermaßen aus:

```
<ProcessManipulationEvent>
   <Action>Replace Model</Action>
   <ProcessType>EDBPM/Import-Warenverteillager</ProcessType>
   <Solution>
      <Item id="oldmodel"><![CDATA[...]]></Item>
      <Item id="newmodel"><![CDATA[...]]></Item>
   </Solution>
   <Condition>
      <Item id="Kunde">Kunde A</Item>
      <Item id="Flug">5888</Item>
   </Condition>
</ProcessManipulationEvent>
```

Dieses Ereignis beinhaltet alle Informationen, die der Prozessmanipulator zu Flexibilisierung benötigt. Die beiden Prozessmodelle sind im `![CDATA[]]`-Feld gekapselt und werden als binäre Daten behandelt.

7.4.2 Prozessmanipulator

Der Prozessmanipulator empfängt das Ereignis und greift auf die enthaltenden Informationen zu. Anhand der Action wird das weitere Verhalten ausgewählt. Da das Condition-Feld mit einem Wert besetzt ist, wird anhand der Bedingung die Menge von zu flexibilisierenden Instanzen ermittelt. In diesem Fall sollen alle Prozesse die Sendungen des Kunden „Kunde A“ des Fluges „5888“ enthalten flexibilisiert werden. Auf diese Weise werden die beiden Instanzen „a0“ und „bf“ als betroffen ermittelt. Die Menge der zu flexibilisierenden Instanzen wird an die Implementierung zum Austausch des Prozessmodells weitergegeben. Die Implementierung ermittelt das Delta zwischen altem und neuem Prozess und generiert aus beiden ein flexibilisiertes Modell. Siehe Abbildung 7.3.

Da die beiden Aktivitäten „Transport-Auftrag zum Zwischenlager“ und „Einlagerung im Zwischenlager“ gelöscht wurden, werden sie für die Instanzen „a0“ und „bf“ unzugänglich gemacht. Zudem werden zwei neue Meta-Sequenzflüsse eingefügt, die die „Anmeldung beim Zoll“ und die „Zoll-Abwicklung“ unter Berücksichtigung der existierenden Bedingungen mit der Aktivität „Transport-Auftrag zum Kunden“ verbinden.

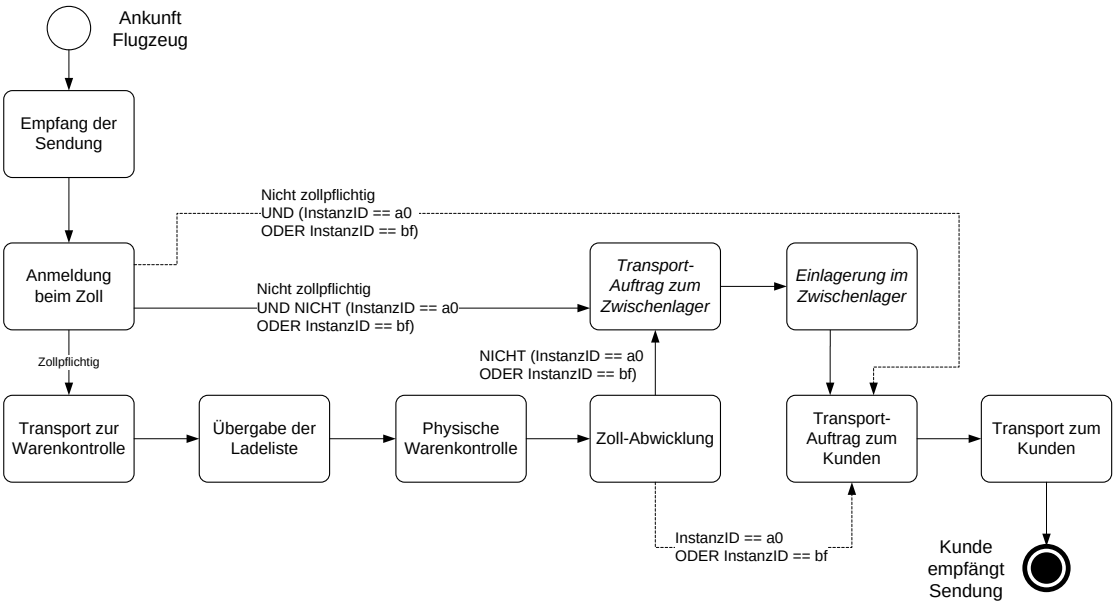

Abbildung 7.3: Flexibilisierter Prozess „Import-Warenverteillager““

Das flexibilisierte Modell wird noch zur Laufzeit der Instanzen neu aufgespielt und so rückwirkend auf die Instanzen angewandt. Der Zustand der Instanzen und deren Position im Prozessablauf wird durch die Prozessausführung selbstständig migriert.

7.4.3 Flexibilisierter Prozess

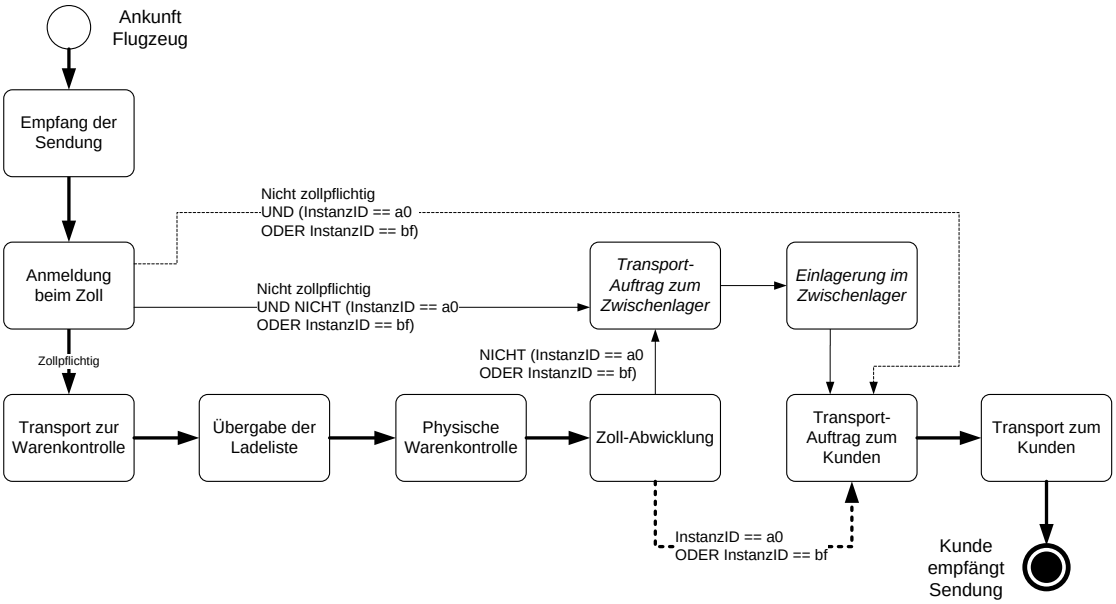

Abbildung 7.4: Prozessinstanz „a0“ im Prozess „Import-Warenverteillager““

In Abbildung 7.4 wird beispielhaft der Ablauf der Instanz „a0“ gezeigt. Da sie zollpflichtig ist, nimmt sie den hervorgehobenen flexibilisierten Weg. Da die Instanz „bf“ in diesem Beispiel nicht-zollpflichtig ist, geht sie direkt von der Anmeldung beim Zoll zum Transport zum Kunden über. Unbetroffene und daher unflexibilisierte Prozessinstanzen verlaufen gänzlich unbetroffen und wählen die Pfade, nicht für die flexibilisierten Instanzen zugänglich sind.

Kapitel 8

Zusammenfassung

8.1 Bewertung des Prototyps

In diesem Abschnitt wird der entwickelte Prototyp anhand der Eigenschaften nach Regev und der elementaren Flexibilisierungs-Features nach Weber bewertet. Im Fazit wird der Prototyp mit den Rahmenbedingungen verglichen, die für ED-BPM im ADiWa-Kontext gestellt wurden.

8.1.1 Abdeckung

Zur Bewertung des entwickelten Prototyps sollen in diesem Abschnitt die implementierten Funktionen mit den am Anfang identifizierten Eigenschaften einer Geschäftsprozessflexibilisierung verglichen werden.

Das Diagramm in Abbildung 8.1 ist eine reduzierte Darstellung des KV-Diagramms 4.2 von Seite 34. Das folgende Diagramm zeigt nur die dort identifizierten ED-BPM-Eigenschaften von Prozessflexibilisierungen im Sinne der Arbeit. Daher sind nur die Eigenschaften und Perspektiven der Instanz-Ebene aufgeführt. Bei einem Prototyp, der Geschäftsprozesse vollständig im Sinne von ED-BPM und ADiWa flexibilisieren könnte, wären alle Felder mit einem × markiert.

In der funktionalen Perspektive sind alle Flexibilisierungen aufgeführt, die erreicht werden können, wenn die Prozessinstanz auf funktionaler Ebene verändert wird. Das heißt, wenn der Prozess in seiner Modellierung verändert werden kann. Die Implementierung von `pm_ReplaceModel` entspricht allen denkbaren funktionalen Flexibilisierungen, da ein Prozessmodell beliebig mit den Mitteln der Modellierung umgestaltet werden kann. Zum jetzigen Zeitpunkt ist es nur durch den webMethods Designer möglich, das veränderte Prozessmodell aufzuspielen. Auf diese Weise können die Instanzen temporär geändert

Abstraktionsebene	**Instanz**	*Subjekt*		*Eigenschaften*			
				inkremental	temporär	unmittelbar	unvorhergesehen
			Funktional	×			×
			Organisatorisch				
			Verhalten	×	×	×	×
			Informationell	×	×	×	
			Operational	×	×	×	

Abbildung 8.1: Abdeckung des Prototyps anhand der Regev-Taxonomie

werden, da dann nur laufende Instanzen vom veränderten Prozessmodell betroffen wären. Weiterhin war es in den Testläufen nicht möglich, laufende Prozessinstanzen in das neue Prozessmodell zu migrieren, weswegen Änderungen nicht unmittelbar sind, sondern erst verzögert für zukünftige Prozessinstanzen gelten. Dennoch können Änderungen unvorhergesehen sein, da die betroffenen Prozessinstanzen nicht auf die Flexibilisierung vorbereitet werden müssen. Sollten die beschriebenen Einschränkungen in Zukunft behoben werden, könnten Prozessflexibilisierungen auf funktionaler Ebene temporär und unmittelbar sein.

Auf organisatorischer Ebene konnten keine Muster identifiziert und implementiert werden, die den Prozess verändert hätten.

Die Verhaltens-Perspektive wird durch die Funktionen zur Steuerung von Prozessinstanzen ausgefüllt. Dazu zählen die Funktionen `pm_Start` und `pm_ControlFlow`. Beide Funktionen arbeiten vollständig auf Instanz-Ebene am bestehenden Prozessmodell. Daher sind sie inkrementell. Zudem sind die Änderungen nur von temporärer Dauer, weil sie strikt Instanz-gekoppelt sind. Sie beeinflussen keine anderen Instanzen in ihrer Ausführung. Der Zugriff auf die Instanzen erfolgt unmittelbar, da Zustands-Änderungen direkt eintreten, und unvorhergesehen, da keine Einstellungen an den Prozessinstanzen notwendig sind, um sie flexibilisieren zu können.

Die informationelle und operationale Perspektive werden beide von der Funktion `pm_Update` implementiert. Durch die Funktion wird zum einen das Daten-Objekt verändert, zum Anderen kann eine Aktivitäts-Implementierung dynamisch referenziert und zur Laufzeit verändert werden. Die Änderungen werden immer auf Instanz-Ebene durchgeführt, wobei inkrementall auf dem bestehenden Daten-Objekt des Prozessmodells aufgebaut wird. Die Änderungen sind temporär, da nur das Daten-Objekt der gewählten Instanz verändert wird. Zudem sind sie unmittelbar, weil die Daten unverzögert verändert werden. Da aber für die Änderung des Daten-Objekts, die Pipeline des Prozesses geloggt werden muss und die Services dynamisch referenziert sein müssen, sind die Änderungen nicht unvorhergesehen, sondern geplant. Ein Prozess muss zuvor darauf vorbereitet werden, später durch die Funktion flexibilisiert zu werden. Dabei handelt es sich jedoch um minimale und immer gleiche Eingriffe zur Design-Zeit des Prozesses. Würde das Daten-Objekt per Definition immer mitgeloggt werden und Services immer dynamisch referenziert werden, könnten Änderungen auf informationeller und operationaler Ebene ebenso unvorhergesehen sein.

Nach dem KV-Diagramm zur Abdeckung des Prototyps unterstützt der Prozessmanipulator vier von fünf Prozess-Perspektiven. Da es bei der organisatorischen Perspektive keine Anhaltspunkte in der vorliegenden Implementierung der Prozessausführungsumgebung gibt, wird die Perspektive nicht unterstützt. Die Verhaltensperspektive stellt dagegen die einzige Perspektive dar, die uneingeschränkt im Sinne von ED-BPM und dieser Arbeit flexibilisiert werden kann. Informationelle und operationale Perspektive teilen sich eine Implementierung, die aber nur eingeschränkt unvorhergesehen flexibilisiert werden kann. Die funktionale Perspektive weist die größten Lücken auf, da hier, mit heutigen Mitteln, nicht das gewünschte Flexibilisierungs-Verhalten erreicht wurde. Das KV-Diagramm zeigt auch, dass die Bewertung der theoretischen Ansätze von ihrer Implementierung abweicht. Der Ansatz zum Austausch des Prozessmodells wurde in Tabelle 5.3 auf Seite 59 noch als temporär und unmittelbar bewertet. Erst in der Implementierung wurden die Einschränkungen aufgedeckt, die bei vorherigen Testläufen an einfachen Beispielen noch nicht auftraten.

8.1.2 Flexibilisierungs-Features

Neben der Bandbreite der möglichen Flexibilisierungen muss der Prototyp auch danach bewertet werden, wie und unter welchen Bedingungen er Flexibilisierungen ausführen kann. Es folgt eine Bewertung anhand der elementaren Flexibilisierungs-Features von Seite 42.

Modell-Versionierung und Instanz-Migration Die Modell-Versionierung und Instanz-Migration ist nur beim Austausch eines Prozessmodells von Interesse, da alle anderen Ansätze strikt auf Instanz-Ebene ablaufen. Die webMethods Process Engine unterstützt die Versionierung von Prozessmodellen und die Migration von Instanzen standardmäßig, wenngleich unzuverlässig und unvorhersehbar. Zum einen ist es möglich, Modelle in verschiedenen Versionen aufzuspielen, von denen dann eine Version für abgeleitete Prozessinstanzen aktiviert werden kann. Die Migration laufender Instanzen einer älteren Version in eine aktive Version ist aber nicht möglich. Theoretisch sollte die Prozessausführungsumgebung dieses Feature unterstützen, da es aber bis zuletzt nicht möglich war, gilt das Feature als nicht erfüllt.

Ad-hoc-Zugriff und Änderungen auf Instanz-Ebene Der Prototyp wurde bewusst auf dieses Feature hin entwickelt, da es auch Teil der Problemstellung dieser Arbeit war. Änderungen auf Instanz-Ebene werden von allen Flexibilisierungen des Prozessmanipulators unterstützt. Da aber nicht alle Flexibilisierungen unvorhergesehen durchgeführt werden können, ist dieses Feature nur teilweise erfüllt.

Konsistenz Bis auf den Ansatz zum Austausch des Prozessmodells bleibt bei jedem der implementierten Ansätze ein konsistentes Prozessmodell zurück. Da bei diesen Ansätzen API-Servies abstrahiert wurden, ist eine interne syntaktische Konsistenz sichergestellt. Die semantische Konsistenz eines Prozessmodells wird hingegen nicht abgesichert. Zum Beispiel ist ein dynamisch referenzierter Service nicht dazu gezwungen die Rückgabewerte in ähnlicher Form zu liefern, wie es der ursprüngliche modellierte Service getan hätte. Daher kann die Konsistenz eines Prozesses auf funktionaler, informationeller, operationaler und Verhaltens-Ebene immer nur syntaktisch gewährleistet werden. Das Feature ist, zumindest für syntaktische Konsistenz, erfüllt.

8.1.3 Fazit

Ein Fazit über den Prototyp lässt sich am besten anhand der Rahmenbedingungen von Seite 32 stellen.

1. *Prozesse sollen auf Instanzebene flexibilisiert werden.*
   Die Prozessinstanzen eines Prozessmodells werden, sofern sie durch die Condition betroffen sind, auf Instanzebene flexibilisiert.

2. *Das Prozessmodell soll für nachfolgende Instanzen unverändert bleiben.*
   Alle Flexibilisierungen laufen, bis auf den Austausch des Prozessmodells, strikt auf Instanzebene und verändern nur die betroffenen Instanzen. Beim Austausch des Prozessmodells wurde ein Weg gefunden, innerhalb eines flexibilisierten Prozessmodells betroffene und unbetroffene Instanzen voneinander zu unterscheiden. Auf diese Weise wird das Prozessmodell nur für betroffene Instanzen verändert.

3. *Prozessinstanzen sollen jederzeit, vollständig und ohne vorherigen Eingriff flexibilisiert werden können.*
   Die Prozessinstanzsteuerung auf Verhaltens-Ebene ist in der Lage, jederzeit, vollständig und ohne vorherigen Eingriff zu flexibilisieren. Die informationellen Flexibilisierungen, die auf ein verändertes Daten-Objekt aufbauen, können nur dann durchgeführt werden, wenn das Daten-Objekt mitgeloggt wird. In diesem Fall müssen, wenngleich minimale, Vorbereitungen getroffen werden. Theoretisch entsprechen auch funktionale Änderungen am Prozessmodell diesen Rahmenbedingungen. Die Änderungen sind vollständig und ohne vorherigen Eingriff möglich. Jedoch sind funktionale Flexibilisierungen praktisch noch nicht möglich, da der Austausch des Prozessmodells nicht zu jedem Zeitpunkt funktioniert.

Die implementierten Flexibilisierungen, unterschiedlicher Art und unterschiedlicher Perspektive, ergeben zusammengenommen einen Prototyp, der einen Großteil der möglichen Geschäftsprozessflexibilisierungen abdeckt. Der Prototyp erfüllt also die Anforderungen und steht der weiteren Entwicklung und Verbesserung offen.

8.1.4 Kritik

Die Überschaubarkeit und Testbarkeit von Software sinkt mit der Anzahl möglicher Zustände. Für den Prototyp haben jeweils CEP-Engine, Prozessausführung und Prozessmanipulator eigene interne Zustände. Durch diese Vielzahl von möglichen Zuständen war es schwierig, den Prototypen als Ganzes zu testen. Schließlich wurde der Prototyp für Tests so erweitert, dass er sich je nach Testfall eigene Ereignisse generierte. Zum Beispiel konnten so zwei Prozessmanipulations-Ereignisse für den Start und Stop einer

Instanz nacheinander gesendet werden. Für das Debugging ein reales Szenario mit Basis-Ereignissen, komplexen Ereignissen, Prozessmanipulator und betroffenen Instanzen zu verwenden wäre zu aufwendig und fehleranfällig gewesen.

Eine der stärksten Erfahrungen bei der Entwicklung des Prototyps waren die Unzulänglichkeiten der verfügbaren Hardware. Der vollständige produktionsähnliche Betrieb einer Unternehmenssoftware in diesem Maßstab und diesen Anforderungen ist nur auf angemessener Hardware möglich. Erst zu einem späten Zeitpunkt konnte der Prototyp auch auf einem neueren System getestet werden, was die Entwicklung deutlich erleichterte. Der Prototyp wurde hauptsächlich innerhalb einer Virtuellen Maschine[1] entwickelt und getestet, deren Anforderungen an den Großteil der verfügbaren Ressourcen des Host-Rechners heranreichten. Selbst wenn die Suite nativ auf den verfügbaren Rechnern betrieben worden wäre, hätte deren Kapazität zu einem sinnvollen Betrieb nicht ausgereicht. Bedingt durch diese mangelnden Ressourcen waren Entwicklung und nachfolgende Test-Läufe sehr zeitintensiv.

Beispielsweise reagierte in der Folge die HTML-basierte Prozesskonsole „My webMethods“ sehr träge und unvorhersehbar. Zum Beispiel wurden Prozesse, die offensichtlich gestartet wurden, nicht angezeigt, da sie von „My webMethods“ noch nicht erfasst werden konnten. Zudem hinderten Bugs der webMethods-Suite die Entwicklung. Bedingt durch die tiefen Eingriffe in das System, stürzte die Software häufig ab oder verfing sich in Endlos-Schleifen. So war es bis zuletzt nicht möglich, ein neues Prozessmodell über den API-Service aufzuspielen, ohne das dies den Integration Server zum Abstürzen gebracht hätte. Der Bug wurde gemeldet aber noch nicht behoben (Stand Februar 2011).

Zur Lösung dieser Probleme wäre ein intensiver Kontakt zu den tatsächlichen Entwicklern der webMethods-Suite wünschenswert gewesen. Dieser war jedoch nur spärlich und indirekt, da webMethods nach wie vor hauptsächlich in den USA und Indien entwickelt wird. Dieser Mangel wird noch deutlicher, wenn man die Kommunikation mit Fragen zur CEP-Engine vergleicht. Der Blick über die Schulter oder der Gang von Büro zu Büro war ergiebiger als zeitzonen-übergreifende Fragen an Entwickler mit für den Verfasser unklaren Zuständigkeiten, die verständlicherweise weder Kontext noch Intention kannten. Die CEP-Engine wird im Gegensatz zu den restlichen webMethods-Komponenten hauptsächlich in Darmstadt entwickelt.

Der Prototyp kann dennoch als Versuch gewertet werden, die gewünschte Funktionalität trotz all dieser Hindernisse zu verwirklichen. Sollten die beschriebenen Fehler der Software beseitigt und auf einem Hardware-starken System ausgeführt werden, steht einem erfolgreichen prototypischen Betrieb nichts im Wege.

[1] Eine Virtuelle Maschine läuft innerhalb einer physischen Maschine ab. Sie verfügt über ein eigenes Betriebssystem und eine virtuelle Festplatte. Die Hardware-Ressourcen werden mit dem ausführenden Rechner (Host) geteilt.

8.2 Zusammenfassung der Arbeit

Das Ziel dieser Arbeit war es, innerhalb des Forschungsprojekts ADiWa eine Möglichkeit zu finden, Geschäftsprozesse dynamisch zu steuern und zu verändern. Diese Flexibilisierung von Geschäftsprozessen sollte ereignis-gesteuert auf der Basis von komplexen Ereignissen aus dem Geschäftsumfeld geschehen. Zur theoretischen Fundierung wurden die Begriffe Complex Event Processing (CEP) und Event-Driven Business Process Management (ED-BPM) eingeführt und erläutert.

Da diese Arbeit in Zusammenarbeit mit der Software AG geschrieben wurde, sollte die Realisierung auf Basis der Produkte der webMethods-Suite geschehen. Im Zusammenhang mit dem ADiWa-Projekt wurde dazu von der Software AG eine eigene CEP-Engine entwickelt. Aufgabe dieser Arbeit war es, eine Verbindung zwischen Ereignis-Quelle und dem bereits etablierten Prozessausführungssystem webMethods Process Engine zu schaffen. Anschließend wurde ermittelt, wie und auf welche weise Prozesse dynamisch flexibilisiert werden können.

Dazu konnten zwei Flexibilitäts-Taxonomien identifiziert werden. Die von den Taxonomien beschriebenen Klassen wurden mit den Anforderungen, die durch das Projekt und durch ED-BPM gestellt wurden, verglichen. Die Flexibilitäts-Klassen wurden über ihre Eigenschaften und den Zeitpunkt ihrer Anwendung ausgewählt. Auf Basis dieses Vergleichs konnten Klassen ermittelt werden, die im Prototyp implementiert werden sollten. Aufgrund der Schonenberg-Taxonomie konnten die Klassen Veränderung, Sequenzfluss und Externalisierung ausgewählt werden. Mithilfe dieser Klassen konnten Flexibilitäts-Muster ermittelt werden, die für die Implementierung vorgesehen sind. Dank der Regev-Taxonomie konnten diese Muster nach der Abstraktionsebene, Umfang und Eigenschaften der Flexibilität klassifiziert und auf vollständige Abdeckung hin überprüft werden.

Die identifizierten Muster wurden im nächsten Schritt dazu benutzt, konkrete Ansätze zu ermitteln, die sich im Kontext von webMethods anboten. Die dort aufgeführten Ansätze wurden theoretisch beschrieben und die spätere Bedeutung für die Implementierung des Prototyps angedeutet. Neben der Instanz-Steuerung, der Veränderung des Datenobjekts und der Änderung im Sequenzfluss wurde an dieser Stelle der Austausch des kompletten Prozessmodells erkannt. Dazu wurden die Probleme und dazugehörige Lösungen beschrieben. Unter anderem gehörten dazu die Probleme, die auftreten, wenn über das Modell rückwirkend Instanzen flexibilisiert werden sollen.

In der anschließenden prototypischen Implementierung wurde der Prozessmanipulator implementiert, der die vorgestellten Ansätze und damit einen Großteil der ermittelten Flexibilitäts-Muster als Funktion bereitstellt. Im Prototyp wurden die Lösungen, die im Zusammenhang mit den Ansätzen entstanden, programmatisch umgesetzt.

Um den Prototypen schließlich ereignis-gesteuert ablaufen zu lassen, wurde die Schnittstelle des Prozessmanipulators zur Integration mit der CEP-Engine entwickelt. Zur Steuerung des Prototyps wurden komplexe Ereignisse definiert, die Flexibilisierungs-Anweisungen enthalten können, die interpretiert und anschließend mithilfe der Prototyp-Funktionalität angewandt werden konnten.

Abschließend wurde der Prototyp bewertet. In dieser Bewertung wurde festgestellt, dass der Prototyp nur einen Teil der anfangs bestimmten Flexibilisierungen bereitstellt. Der Austausch des Prozessmodells und die Veränderung von Daten-Objekten funktionieren zwar theoretisch, aber aufgrund von technischen Hindernissen nicht praktisch.

Im Prototyp wurde versucht, für eine bereits existierende Prozessausführungsumgebung nachträglich Prozessflexibilisierung zur Laufzeit zu implementieren. Dabei bestand die Process Engine als Blackbox, auf die anhand von API-Services zugegriffen wurde. Dabei wäre es günstiger gewesen, innerhalb der Process Engine zu arbeiten und nicht außerhalb. Auf die Weise konnte immer nur ein gewünschtes Verhalten erwartet und niemals direkt herbeigeführt werden. Zum Beispiel wäre es einfacher, eine Instanz direkt um eine hinzugefügte Aktivität zu erweitern, statt diesen Ablauf indirekt über ein aufgespieltes Prozessmodell erreichen zu wollen.

Der Prototyp und die im Verbund entstandenen Überlegungen können als gelungen betrachtet werden, da sie für die webMethods-Suite und Geschäftsprozessflexibilisierung im Allgemeinen Überlegungen bereitet haben. Als Erkenntnis der Entwicklung des Prototyps empfiehlt der Verfasser dieser Arbeit, die Prozessflexibilität zur Ausführungszeit als direkten Bestandteil der Prozessausführungsumgebung zu implementieren. Eine integrierte Prozessflexibilisierungs-Lösung könnte die Migration und Konsistenz von Instanzen kapseln und dem Prozessmanipulator transparent offenlegen. Auf diese Weise könnten Instanzen direkt durch Hinzufügen, Löschen oder Bearbeiten verändert werden und das anschließende Instanz-Verhalten direkt herbeigeführt und damit erwartet werden. Andernfalls besteht ein zu hohes Risiko für inkonsistente Prozessinstanzen und für unvorhersagbare Laufzeitfehler im weiteren Prozessverlauf.

8.3 Ausblick

In der Arbeit konnten nicht alle der offenen Probleme und Fragen geklärt werden. Die Einschränkungen, wie sie beim Prozessmodell-Austausch und dem Verändern des Daten-Objekts aufgetreten sind, müssen für eine vollständige Abdeckung durch den Prototyp aufgehoben werden. Dazu würde die Behebung der beschriebenen Bugs beitragen, sowie die korrekte Implementierung der Pipeline beim Ändern des Daten-Objekts. Diese beiden Schritte haben hohe Priorität, da sie den Prototyp erheblich verbessern würden.

Zudem betrachtet der Prototyp nach bisherigem Stand die Änderungen an Prozessen nur auf syntaktischer Ebene. Neben der Konsistenzprüfung des Prozessmodells und der Änderungen sollte überprüft werden, ob Flexibilisierungen Transaktionen oder Abhängigkeiten, z. B. bei Eingangs-Daten, verletzen. Ob eine derartige Lösung programmatisch gelöst werden kann, sei dahingestellt, weil die Änderungen jeweils starkes Kontextwissen über den Prozess voraussetzen. Aktuell setzt der Prozessmanipulator dieses Kontextwissen voraus, da er selbst nicht wissen kann, ob die Zoll-Aktivität gelöscht werden darf oder nicht.

Außerdem beschränkt sich der Prototyp bisher immer auf ein Prozessmodell und verändert nur Instanzen dieses Modells. In der Praxis kann aber mehr als ein Prozessmodell von einem komplexen Ereignis betroffen sein. Zum Beispiel wird von der Verspätung einer Sendung sowohl das Warenverteillager als auch die Rechnungsabteilung betroffen sein, die den schnellen Transport zum Kunden anders verbuchen muss als die übliche Zwischenlagerung.

Anhand der Schonenberg-Taxonimie wurde bereits die Flexibilitäts-Klasse der „Unterspezifizierung“ vernachlässigt, da sie geplante Vorbereitungen vorausgesetzt hätte. In webMethods können aber Sub-Prozesse von Haus aus dynamisch eingebunden werden, daher sollte dieser Flexibilitäts-Ansatz noch näher betrachtet werden.

In der Arbeit wurden elementare Features nach Weber genannt, die eine flexible Prozessausführung auf jeden Fall unterstützen sollte. Weber definiert zusätzlich erweiternde Features, die für die reine Implementierung von Prozessflexibilität nicht zwingend notwendig sind. Sie sind jedoch speziell im Hinblick auf eine zukünftige Entwicklung im produktiven Umfeld interessant. [WRRM-2008]

8.3.1 Erweiternde Features der Prozessausführung

Als erweiterndes Feature sollte die Flexibilisierung in der Lage sein, den Zugriff auf bestimmte Prozesselemente durch eine *Zugriffsbeschränkung*[2] zu verweigern. Bestimmte Elemente eines Prozessmodells, die als kritisch gelten, sollen so von der Flexibilisierung ausgeschlossen werden können. Das sind z. B. Elemente, die elementare Dinge der Geschäftslogik repräsentieren, die auf keinen Fall verändert werden dürfen.

Flexibilisierungen betreffen in vielen Fällen eine große Anzahl von Instanzen, was die *Wiederverwendbarkeit*[3] erhöht. Die Flexibilisierung wird dann für all diese Instanzen durchgeführt. Wenn eine solche Änderung aber nur temporär ist, bleiben alle zukünftigen Instanzen von ihr unbetroffen. Es kann aber zu einem späteren Zeitpunkt notwendig sein, eine bestimmte Flexibilisierung erneut durchzuführen. Eine Flexibilisierung zu modellieren und alle daraus entstehenden Probleme zu berücksichtigen kann viel Expertenwissen erfordern. Aus diesem Grund soll es möglich sein, bereits durchgeführte Flexibilisierungen in einem Pool zu sammeln, um sie zu einem späteren Zeitpunkt erneut durchführen zu können.

Zur *Analyse und Diagnose*[4] bietet es sich zudem an, die Gründe und Ergebnisse von Flexibilisierungen zu sammeln, um sie auszuwerten. Auf diese Weise können möglicherweise Häufungen gefunden werden. Mithilfe der erkannten Häufungen kann das Prozessmodell evolutionär verändert werden [WRR-2007b, Seite 21]. Aus dieser Analyse lassen sich Flexibilisierungs-Häufungen erkennen. Wenn ein Prozess oft nach einem bestimmten Muster flexibilisiert wird, kann darüber nachgedacht werden die Flexibilisierungen in das Prozessmodell einzupflegen. Auf diese Weise kann der Prozess durch Prozessverbesserung und Prozessinnovation evolutionär flexibilisiert werden.

8.3.2 Erweitertes ED-BPM

Unter den Begriff „Erweitertes ED-BPM“ könnten die Eigenschaften der Prozessflexibilisierung *revolutionär*, *permanent*, *verzögert* und *geplant* fallen, die anhand des KV-Diagramms auf Seite 33 vernachlässigt wurden.

Dazu zählen zum Beispiel *revolutionäre* Änderungen, die aufgrund von Ereignissen der Vergangenheit ein neues Prozessmodell erstellen. Wäre die Flexibilisierung zudem *permanent*, hätten die Flexibilisierungen der Vergangenheit auch eine Auswirkung auf zukünftige Ereignisse. *Verzögerte* Flexibilisierungen wären ein Gegenentwurf zu Ad-hoc-Zugriffen

[2] Access Control for Changes
[3] Change Reuse
[4] Traceability and Analysis

und würden nur spätere Instanzen verändern. *Geplante* Flexibilisierungen würden gewisse Stellen definieren, an denen ein Prozess flexibilisiert werden kann, um die Integrität eines Prozesses zu sichern.

Auf *organisatorischer* Ebene könnten z. B. die Zugriffsberechtigten für einen Prozess flexibilisiert werden. Zum Beispiel könnte so für menschliche Aktivitäten beim Ausfall eines Mitarbeiters flexibel ein anderer Mitarbeiter für die Aktivität ausgewählt und zugewiesen werden.

Die Trennung zwischen den Prozess-Abstraktionsebenen Instanz und Modell erscheint schwieriger. In dieser Arbeit sollten Prozesse nur auf Instanzebene flexibilisiert werden. Eine Flexibilisierung auf *Modellebene* käme einer Prozessadaption oder Prozessinnovation gleich. Dabei stellt sich die Frage, ob solch umfangreiche und sensible Änderungen überhaupt ereignis-gesteuert durchgeführt werden sollten. Bei Flexibilisierungen des Prozessmodells erscheint der klassische Weg der direkten evolutionären Modellierung von Hand besser, da Prozessmodelle auf Erfahrungen und Richtlinien basieren. Eine *ereignisgesteuerte Prozessmodellierung zur Design-Zeit* wäre am ehesten mit neuronalen Netzen zu vergleichen. Ob sich Geschäftsabläufe auf diese Weise abbilden lassen, ist fragwürdig.

Abkürzungsverzeichnis

BPM	Business Process Management
BPMN	Business Process Management Notation
BPMS	Business Process Management System
CEP	Complex Event Processing
EDA	Event-Driven Architecture
EDBPM	Event-Driven Business Process Management
EPL	Event-Processing Language
ESB	Enterprise Service Bus
IS	Integration Server
JMS	Java Message Service
MOF	Meta-Object Facility
RFID	Radio-Frequency Identification
SAG	Software AG
SOA	Service-oriented Architecture
WM	webMethods
XML	Extensible Markup Language

Literaturverzeichnis

[ADi-2009a] ADiWa: *Allianz digitaler Warenfluss - Antrag auf Förderung für eine IKT2020-Leitinnovation.* 2009

[ADi-2009b] ADiWa: *Allianz Digitaler Warenfluss - Über das Projekt.* `http://www.ADiWa.net/index.php?id=125`. Version: Januar 2009, Abruf: 10.02.2011

[ADi-2010] ADiWa: *Arbeitspaket G5 Geschäftsprozessausführung und -controlling - Konsolidierte Anforderungen.* 2010

[AEE+-2009a] Ammon, R. von ; Emmersberger, C. ; Ertlmaier, T. ; Etzion, O. ; Paulus, T. ; Springer, F.: *Existing and future standards for event-driven business process management.* In: *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems* ACM, 2009, S. 24

[AEE+-2009b] Ammon, R. von ; Ertlmaier, T. ; Etzion, O. ; Kofman, A. ; Paulus, T.: *Integrating Complex Events for Collaborating and Dynamically Changing Business Processes.* In: ICSOC/ServiceWave (2009), S. 23–24

[AESW-2008] Ammon, Rainer von ; Emmersberger, Christoph ; Springer, Florian ; Wolff, Christian: *Event-Driven Business Process Management and its Practical Application Taking the Example of DHL.* In: Future Internet Symposium (2008)

[All-2009] Allweyer, Thomas: *BPMN 2.0-Business Process Model and Notation.* Norderstedt : Books on Demand, 2009

[BD-2010] Bruns, Ralf ; Dunkel, Jürgen: *Event-Driven Architecture: Softwarearchitektur für ereignis-gesteuerte Geschäftsprozesse.* München : Springer, 2010

[BPM-2011] *Business Process Modell Notation (BPMN) Version 2.0, Spezifikation.* `http://www.omg.org/spec/BPMN/2.0/PDF`. Version: Januar 2011, Abruf: 10.02.2011

[Bun-2008] Bundesministerium für Bildung und Forschung: *Digitales Produktgedächtnis macht Warenfluss sicher und schnell.* `http://www.bmbf.de/press/2286.php`. Version: April 2008, Abruf: 10.02.2011

[Bus-2010] Businessweek: *Software AG aquires Real Time Monitoring to keep expanding.* `http://www.businessweek.com/news/2010-04-27/software-ag-acquires-real-time-monitoring-to-keep-expanding.html`. Version: April 2010, Abruf: 10.02.2011

[Cha-2004] Chappell, David A.: *Enterprise Service Bus: Theory in Practice.* Sebastopol, Kalifornien : O'Reilly Media, Inc., 2004

[CS-2009] Chandy, K ; Schulte, R: *Event Processing: Designing IT Systems for Agile Companies.* 1. New York : McGraw-Hill Osborne Media, 2009

[DRRM-2010] Dadam, P. ; Reichert, M. ; Rinderle-Ma, S.: *Prozessmanagementsysteme: Nur ein wenig Flexibilität wird nicht reichen.* In: Informatik-Spektrum (2010)

[DZG-2010] Döhring, Markus ; Zimmermann, Birgit ; Godehardt, Eicke: *Extended Workflow Flexibility using Rule-Based Adaptation Patterns with Eventing Semantics.* In: *Informatik2010 Service Science, Workshop: Business-Rule basierte Servicesteuerung.* Leipzig : Gesellschaft für Informatik, September 2010

[Etz-2010] Etzion, Opher: *Event Processing Thinking - Where does EDBPM term come from?* `http://epthinking.blogspot.com/2010/12/where-does-edbpm-term-come-from.html`. Version: Dezember 2010, Abruf: 10.02.2011

[For-2009] Forrester: *The Forrester Wave: Complex Event Processing (CEP) Platforms.* 2009

[HCD+-1994] Hammer, M. ; Champy, J. ; Daniels, A.C. ; James, P.O.S. ; Hughes, A.M.: *Reengineering the corporation.* New York : Harper Business, 1994

[HSK-2004] Harrländer, Nico ; Schönherr, Marten ; Krallmann, Hermann: *Flexibilisierung druch integrierte prozessorientierte IT-Systeme.* In: Erfolgsfaktor Flexibilität: Strategien und Konzepte für wandlungsfähige Unternehmen (2004)

[Kr2007] Krämer, Jürgen: *Continuous Queries over Data Streams-Semantics and Implementation*, Universität Marburg, Diss., 2007

[Luc-2007] Luckham, David: *SOA, EDA, BPM and CEP are all Complementary.* `http://complexevents.com/wp-content/uploads/2007/07/Soa_EDA_Part2.pdf`. Version: 2007, Abruf: 10.02.2011

[MAR-2008] Mulyar, N. ; Aalst, W. van d. ; Russell, N.: *Process Flexibility Patterns.* In: BETA Working Paper Series, WP 251, Eindhoven University of Technology (2008)

[Ora-2010] Oracle: *The Instantly Responsive Enterprise: Integrating Business Process Management and Complex Event Processing.* In: An Oracle White Paper (2010)

[Rea-2010] Realtime Monitoring RTM: *RTM Analyzer - An infrastructure for tailored Complex Event Processing Solutions.* `http://www.realtime-monitoring.de/pdfs/WhitePaper_RTM_Analyzer_EN.pdf`. Version: Januar 2010, Abruf: 10.02.2011

[RSS-2006] Regev, G. ; Soffer, P. ; Schmidt, R.: *Taxonomy of Flexibility in business processes.* In: *Proceedings of the 7th Workshop on Business Process Modelling, Development and Support (BPMDS'06).* Luxemburg : Presses universitaires de Namur, 2006

[RW-2005] Regev, G. ; Wegmann, A.: *A regulation-based view on business process and supporting system flexibility.* In: *Proceedings of the CAiSE'05* Bd. 5. Riga : FEUP, 2005, S. 91–98

[Sch-2011] Schulte, Roy: *Event Processing in Business Applications.* `http://complexevents.com/slides/Gartner_Schulte.ppt`. Version: Januar 2011, Abruf: 10.02.2011

[SMR$^+$-2007] Schonenberg, H. ; Mans, R. ; Russell, N. ; Mulyar, N. ; Aalst, W. van d.: *Towards a taxonomy of process flexibility (extended version).* In: BPM Center Report BPM-07-11, BPMcenter. org (2007)

[SMR$^+$-2008] Schonenberg, H. ; Mans, R. ; Russell, N. ; Mulyar, N. ; Aalst, W. van d.: *Process flexibility: A survey of contemporary approaches.* In: Advances in Enterprise Engineering I (2008), S. 16–30

[Sof-2009a] Software AG: *Administering webMethods Integration Server*, Dezember 2009. `http://documentation.softwareag.com/webmethods/wmsuite8_ga/Integration_Server_and_Process_Engine/8-0-SP1_Administering_Integration_Server.pdf`, Abruf: 10.02.2011

[Sof-2009b] Software AG: *Administering webMethods Process Engine*, Dezember 2009. http://documentation.softwareag.com/webmethods/wmsuites/wmsuite8_ga/Integration_Server_and_Process_Engine/8-0-SP1_Administering_Process_Engine.pdf, Abruf: 10.02.2011

[Sof-2009c] Software AG: *Understanding the webMethods Product Suite*, Januar 2009. http://documentation.softwareag.com/webmethods/wmsuite8_ga/Cross_Product/8-0_Understanding_webMethods_Product_Suite.pdf, Abruf: 10.02.2011

[Sof-2009d] Software AG: *webMethods Integration Server Built-In Services Reference*, Dezember 2009. http://documentation.softwareag.com/webmethods/wmsuites/wmsuite8_ga/Integration_Server_and_Process_Engine/8-0-SP1_Integration_Server_Built-In_Services_Reference.pdf, Abruf: 10.02.2011

[Sof-2009e] Software AG: *webMethods Monitor Built-In Services Reference*, Dezember 2009. http://documentation.softwareag.com/webmethods/wmsuites/wmsuite8_ga/Optimize_and_Monitor/8-0-SP1_Monitor_Built-In_Services_Reference.pdf, Abruf: 10.02.2011

[Sof-2009f] Software AG: *webMethods Monitor User Guide*, Dezember 2009. http://documentation.softwareag.com/webmethods/wmsuites/wmsuite8_ga/Optimize_and_Monitor/8-0-SP1_Monitor_Users_Guide.pdf, Abruf: 10.02.2011

[Sof-2010] Software AG: *Company History.* http://www.softwareag.com/Corporate/Company/companyinfo/history/default.asp. Version: Dezember 2010, Abruf: 10.02.2011

[Wes-2007] Weske, Mathias: *Business Process Management: Concepts, Languages, Architectures.* Berlin : Springer, 2007

[WRR-2007a] Weber, B. ; Rinderle, S. ; Reichert, M.: *Change patterns and change support features in process-aware information systems.* In: *Proceedings of the 19th international conference on Advanced information systems engineering.* Trondheim : Springer, 2007, S. 574–588

[WRR-2007b] Weber, B. ; Rinderle, S. ; Reichert, M.: *Identifying and evaluating change patterns and change support features in process-aware information systems* / University of Twente. Enschede, 2007. – Forschungsbericht

[WRRM-2008] Weber, B. ; Reichert, M. ; Rinderle-Ma, S.: *Change patterns and change support features-enhancing flexibility in process-aware information systems.* In: Data & knowledge engineering 66 (2008), Nr. 3, S. 438–466

Anhang

Prototyp-Methoden

`public final class processManipulator_SVC`
Die Prozessmanipulator-Klasse. Siehe Abschnitt 6.1 auf Seite 72.

`public static final void processManipulator`
`(IData pipeline)`
Die einzige Methode der Prozessmanipulator-Klasse. Ihre Aufgabe ist es, die Methode `pm_ProcessManipulator` zu starten, Debug-Szenarios auszuwählen und abzubrechen, wenn ein anderes Ereignis als das Prozessmanipulations-Ereignis den Prozessmanipulator startet.

`private static String debug_getDebugProcessManipulationEvent`
`(Integer scenario)`
Eine Methode zum Debugging, die je nach Szenario unterschiedliche Prozessmanipulations-Ereignisse zum Testen liefert

---

`private static IData pm_ProcessManipulator`
`(IData pipeline, Integer scenario)`
Controller-Klasse für den Prozessmanipulator zur Auswahl der Flexibilitäts-Funktion und Übergabe der Parameter. Siehe Abschnitt 6.2 auf Seite 74.

`private static void pm_Redo`
`(String action, String processtype, String solution, String condition)`
Wiederholung von Aktivitäten. Siehe Abschnitt 6.7 auf Seite 85.

`private static void pm_List`
`(String action, String processtype, Map<String, String> solutions, Map<String, String> conditions)`
Ausgabe der Instanz-Daten. Siehe Abschnitt 6.3 auf Seite 75.

`private static void pm_UpdateDataObject`
`(String action, String processtype, Map<String, String> solutionMap, Map<String, String> condition)`
Änderungen am Datenobjekt. Siehe Abschnitt 6.8 auf Seite 86.

`private static void pm_NewActivityInstance`
`(String action, String processtype, String solution, String condition)`
Zusätzliche Aktivitäts-Instanzen. Siehe Abschnitt 6.7 auf Seite 85.

`private static void pm_ReplaceModel`
`(String action, String processtype, String oldmodel, String newmodel, Map<String, String> conditionMap)`
Prozessmodell-Austausch. Siehe Abschnitt 6.6 auf Seite 79.

`private static void pm_ControlFlow`
`(String action, String processtype, String solution, String condition)`
Prozesssteuerung laufender Instanzen. Siehe Abschnitt 6.5 auf Seite 78.

`private static void pm_Start`
`(String action, String processtype, Map<String, String> solutionMap, String condition)`
Start einer neuen Prozessinstanz. Siehe Abschnitt 6.4 auf Seite 77.

`private static List<IData> tool_getInstancesFromCondition`
`(String processtype, Map<String, String> conditionMap)`
Liefert alle Instanzen eines `ProcessTypes` die den Bedingungen der Condition-Map entsprechen. Siehe Abschnitt 6.3 auf Seite 75.

`private static void tool_submitDocFromXML`
`(String documentTypeName, String xmldata)`
Publiziert ein Dokument nach dem übergebenen XML-Dokument. Siehe Abschnitt 6.4 auf Seite 77.

`private static void tool_resubmit`
`(IData instance, String stepID, String stepPipeline)`
Wiederholt eine Prozessinstanz am Step mit der übergebenen `stepID` unter Verwendung der `stepPipeline`. Siehe Abschnitt 6.7 auf Seite 85.

`private static void tool_changeInstanceStatus`
`(String instanceID, String modelID, String status)`
Die Methode versetzt eine beliebige Instanz in den übergebenen Status. Siehe Abschnitt 6.5 auf Seite 78.

`private static void tool_suspendInstance`
`(String instanceID, String modelID)`
Methode zum Pausieren von Instanzen unter Verwendung von `tool_changeInstanceStatus`.

`private static void tool_resumeInstance`
`(IData instance)`
Methode zum Wiederaufnehmen von Instanzen unter Verwendung von `tool_changeInstanceStatus`.

`private static void tool_stopInstance`
`(IData instance)`
Methode zum Stoppen von Instanzen unter Verwendung von `tool_changeInstanceStatus`.

`private static List<IData> tool_getInstancesThatMatchValue`
`(String modelName, String field, String value)`
Liefert alle Instanzen die eine in einem übergebenen Feld den übergebenen Wert haben. Die Methode vergleicht alle Werte einer jeden Instanz mit den `field-value`-Paaren. Hat eine Instanz einen enstprechenden Wert, wird die Instanz der Rückgabe-Liste hinzugefügt. Durch Mehrfach-Abfragen lässt sich Schnittmenge der Ergebnis-Menge bestimmen.

`public static void tool_listInstance`
`(IData instance)`
Gibt Daten einer Prozessinstanz aus. Siehe Abschnitt 6.3 auf Seite 75.

`public static IData[] tool_getInstances`
`(String modelName)`
Liefert alle Instanzen eines Prozessmodells. Dabei wird keine Rücksicht auf den Zustand der Instanz genommen. In der Menge befinden sich auch auch bereits beendete Instanzen.

`public static IData[] tool_getSteps`
`(String modelID)`
Liefert alle Steps eines Prozessmodells. Die Methode wird dazu benutzt, um vom Aktivitätsnamen auf eine Step-ID zu schließen. Step-IDs dienen der internen Repräsentation.

`public static IData[] tool_getCustomData`
`(String instanceID)`
Liefert das Daten-Objekt einer Instanz.

`public static int tool_getModelVersion`
`(String model)`
Liefert die Version eines Prozessmodells, das als String übergeben wurde.

`public static void tool_deployProcessModel`
`(String name, String project, String model, int version)`
Spielt ein Prozessmodell in einer bestimmten Version in die Prozessausführung auf. Siehe Unterabschnitt 6.6.4 auf Seite 85.

`private static String tool_getStringValue`
`(final IData input, final String elementName)`
Gibt den Wert für das übergebene Element eines IData-Objekts als String zurück. Die Methode kapselt die sehr aufwendigen Abfragne von Werten in IData-Objekten.

`private static Object tool_getValues`
`(final IData input, final String elementName)`
Gibt ein oder mehrere Werte für das übergebene Element eines IData-Objekts zurück. Das Objekt muss anschließend je nach erwarteten Datentyp in ein IData-Objekt, String oder Integer gecastet werden. Die Methode kapselt ebenso die aufwendigen IData-Daten-Abfragen.

`public static final String flex_flexProcess`
`(String oldModel, String newModel, List<String> flexInstances)`
Methode um ein Delta aus einem alten (`oldModel`) und veränderten (`newModel`) Prozessmodell zu erstellen. Siehe Unterabschnitt 6.6.3 auf Seite 82.

`private static void flex_addConditions`
`(Element t, String logic, List<String> flexInstances)`
Die Methode erweitert den übergebenen Sequenzfluss `t` um die Meta-Bedingungen bezüglich der Instanzen aus flexInstances. In `logic` ist die Logik in der Form „!=“ oder „==“ definiert. Das Element `t` ist ein JDOM-Objekt[5], dem pro Bedingung ein Kindelement hinzugefügt wird.

[5]Eine Library zum Parsen von XML-Dokumenten in Java. `http://jdom.org`

`private static List<Element> flex_getTransitionsFromUID`
`(String uid, List<Element> transitions)`
Liefert alle Sequenzflüsse die vom Element mit der übergebenen ID wegführen. Damit können diese beim Modifzieren eines Elements dupliziert werden. Die Liste der enthält JDOM-Objekte aller potentiellen Sequenzflüsse.

`private static List<Element> flex_getTransitionsToUID`
`(String uid, List<Element> transitions)`
Liefert alle Sequenzflüsse die zu Element mit der übergebenen ID hinführen. Damit können diese beim Modifzieren eines Elements dupliziert werden. Die Liste der enthält JDOM-Objekte aller potentiellen Sequenzflüsse.

`private static List<Element> flex_getChangedElements`
`(List<Element> oldelements, List<Element> newelements)`
Liefert alle die Elemente die im veränderten Modell verändert wurden. Siehe Unterabschnitt 6.6.3 auf Seite 82.

`private static List<Element> flex_getUnchangedElements`
`(List<Element> oldelements, List<Element> newelements)`
Liefert alle die Elemente die im veränderten Modell nicht verändert wurden. Siehe Unterabschnitt 6.6.3 auf Seite 82.

Beispiel Prozessmanipulations-Ereignisse

Prozess-Steuerung

```
<ProcessManipulationEvent>
    <Action>Create</Action>
    <ProcessType>EDBPM/ImportWarenverteillager</ProcessType>
    <Solution>
        <Item id='Sendung'>5888</Item>
        <Item id='Sender'>Trade Company A</Item>
        <Item id='Herkunft'>Hong Kong</Item>
        <Item id='Kunde'>Kunde A</Item>
        <Item id='Adresse'>64283 Darmstadt</Item>
        <Item id='Ladeliste'>...</Item>
    </Solution>
    <Condition></Condition>
</ProcessManipulationEvent>
```

```
<ProcessManipulationEvent>
    <Action>Stop</Action>
    <ProcessType>EDBPM/ImportWarenverteillager</ProcessType>
    <Solution></Solution>
    <Condition>Running</Condition>
</ProcessManipulationEvent>
```

```
<ProcessManipulationEvent>
    <Action>Resume</Action>
    <ProcessType>EDBPM/ImportWarenverteillager</ProcessType>
    <Solution></Solution>
    <Condition>Paused</Condition>
</ProcessManipulationEvent>
```

Ausgabe der Prozessdaten

```
<ProcessManipulationEvent>
    <Action>List</Action>
    <ProcessType>EDBPM/ImportWarenverteillager</ProcessType>
    <Solution></Solution>
    <Condition>
        <Item id='Kunde'>Kunde A</Item>
    </Condition>
</ProcessManipulationEvent>
```

Prozessmodell austauschen

```
<ProcessManipulationEvent>
   <Action>Replace Model</Action>
   <ProcessType>EDBPM/Import-Warenverteillager</ProcessType>
   <Solution>
      <Item id="oldmodel"><![CDATA[...]]></Item>
      <Item id="newmodel"><![CDATA[...]]></Item>
   </Solution>
   <Condition>
      <Item id="Kunde">Kunde A</Item>
      <Item id="Flug">5888</Item>
   </Condition>
</ProcessManipulationEvent>
```

Daten-Objekt verändern

```
<ProcessManipulationEvent>
    <Action>Update</Action>
    <ProcessType>EDBPM/ImportWarenverteillager</ProcessType>
    <Solution>
        <Item id='Sender'>Trade Company B</Item>
        <Item id='Herkunft'>Republic China</Item>
        <Item id='Kunde'>Kunde B</Item>
        <Item id='Adresse'>36179 Bebra</Item>
    </Solution>
    <Condition>
        <Item id='Sendung'>5888</Item>
    </Condition>
</ProcessManipulationEvent>
```

```
<ProcessManipulationEvent>
    <Action>Update</Action>
    <ProcessType>EDBPM/ImportWarenverteillager</ProcessType>
    <Solution>
        <Item id='Service1'>Service2</Item>
    </Solution>
    <Condition>
        <Item id='Service1'>Service1</Item>
        <Item id=''>Paused</Item>
    </Condition>
</ProcessManipulationEvent>
```

Wiederholung

```
 <ProcessManipulationEvent>
    <Action>New Activity Instance</Action>
    <ProcessType>EDBPM/ImportWarenverteillager</ProcessType>
    <Solution>Anmeldung beim Zoll</Solution>
    <Condition></Condition>
</ProcessManipulationEvent>
```

```
<ProcessManipulationEvent>
    <Action>Redo</Action>
    <ProcessType>EDBPM/ImportWarenverteillager</ProcessType>
    <Solution>Zoll-Abwicklung</Solution>
    <Condition>
        <Item id='Kunde'>Kunde A</Item>
    </Condition>
</ProcessManipulationEvent>
```