

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

Diplomarbeit Nr. 2898

Enforcement auf laufenden BPEL-Prozessen

Mattanja Kern

Studiengang: Softwaretechnik
Prüfer: Prof. Dr. Frank Leymann
Betreuer: Dipl.-Inf. Daniel Schleicher

begonnen am: 02. März 2009
beendet am: 01. September 2009

CR-Klassifikation: D.2.2, D.3.3, H.4.1

Inhaltsverzeichnis

1	Einleitung	7
1.1	Motivation und Ziel der Arbeit	8
1.2	Aufbau der Arbeit	8
1.3	Typographische und formale Konventionen	9
2	Grundlagen	10
2.1	Geschäftsprozesse	10
2.2	Workflow-Modelle	10
2.3	Workflow-Instanzen	11
2.4	BPEL	11
2.4.1	Struktur der Sprache	12
2.4.2	Scopes	12
2.4.3	Variablen	13
2.4.4	Aktivitäten	13
2.4.5	Link (Übergang)	15
2.4.6	Transition Condition (Übergangsbedingung)	15
2.4.7	Join Condition (Zusammenführungsbedingung)	16
2.4.8	Dead-Path-Elimination	16
2.5	Workflow Engines	16
2.6	Apache ODE	17
2.6.1	Management API	17
2.7	Beispielprozesse	18
3	Workflowmodifikation	21
3.1	Workflow-Schema-Änderungen	22
3.2	Workflow-Instanz-Änderungen	23
3.2.1	Zustand einer Prozessinstanz	24
3.2.2	Zustand einer Aktivität	25

3.2.3	<i>Modification trace</i> und <i>Execution trace</i>	25
3.2.4	Modifikation in Schleifen	26
3.3	Gültigkeit der Modifikation	28
3.3.1	Gültigkeitsbereiche von Variablen	28
3.3.2	Abhängigkeiten im Kontrollfluss	28
3.3.3	Abhängigkeiten im Datenfluss	29
3.3.4	Überprüfung der Änderung	29
3.3.5	Prozessesemantik	30
3.3.6	Verwandte Arbeiten	30
3.4	Änderungsmuster	30
3.5	Operationen im Detail	32
3.5.1	Einfügen	32
3.5.2	Entfernen	37
3.5.3	Verschieben	42
3.5.4	Ersetzen	44
3.5.5	Vertauschen	44
3.5.6	Wiederholen	45
4	Konzepte zur Realisierung	46
4.1	Instanzmodifikation durch Instanzmigration	46
4.2	Instanzmodifikation durch Replay	48
4.3	Eventbasierte Instanzmodifikation	50
4.4	Vergleich der Verfahren	51
4.4.1	Modifikation aller Instanzen (<i>Migrate</i>)	51
4.4.2	Modifikation einzelner Instanzen (<i>Adapt</i>)	52
4.4.3	Unterbrechung der Prozessausführung	52
4.4.4	Partner links	53
4.4.5	Modifikation bereits durchgeführter Aktivitäten	53
4.4.6	Zusammenfassung des Vergleiches	53
5	Entwurf & Implementierung	55
5.1	Architektur	55
5.1.1	Status einer Prozessinstanz	55
5.1.2	Eventmodell & Zustand einer Aktivität	56
5.1.3	Definition einer neuen Aktivität	58
5.1.4	Durchführung der Modifikation	59
5.1.5	Fault Handling	60
5.1.6	Persistenz der Änderungen	61
5.2	Webservice	61

5.2.1	Identifikation der Prozessinstanz	61
5.2.2	Identifikation einer Aktivität	62
5.2.3	Datentyp zur Lokation der Modifikation	62
5.2.4	Operation zum Einfügen einer Aktivität	65
5.2.5	Operation zum Entfernen einer Aktivität	65
5.2.6	Operation zum Modifizieren von Variablen	66
6	Anwendungsbeispiel	67
7	Zusammenfassung und Ausblick	71
7.1	Ausblick	71
A	Codelistings	73
B	Begriffserklärung	83
	Literaturverzeichnis	85

Abbildungsverzeichnis

2.1	Bestellprozess (Purchase Order Process) [OASo7]	18
2.2	Vereinfachtes Beispiel eines Kreditprozesses vor und nach dem Hinzufügen einer Aktivität	20
3.1	Zustände einer Prozessinstanz [LRoo]	24
3.2	Zustände und Zustandsübergänge einer Aktivität [LRoo]	26
3.3	<i>Execution trace</i> und <i>Modification trace</i>	27
3.4	Seriell Einfügen einer Aktivität	33
3.5	Paralleles Einfügen einer Aktivität	34
3.6	Einfügen einer Aktivität unter Beibehaltung der <i>joinCondition</i>	38
3.7	Entfernen einer Aktivität	38
3.8	Entfernen einer parallel verlaufenden Aktivität	42
3.9	Entfernen einer zusammenführenden Aktivität	43
3.10	Verschieben einer Aktivität	43
3.11	Vertauschen zweier Aktivitäten	44
4.1	Darstellung der Instanzmodifikation durch Instanzmigration	47
4.2	Ablauf der eventbasierten Modifikation	50
5.1	Vorgehen beim Entfernen einer Aktivität	60
5.2	Übersicht über den Webservice <i>ModificationService</i>	61
5.3	Darstellung des Datentyps <i>ActivityLocationType</i>	63
5.4	Darstellung des Datentyps <i>InstanceListType</i>	63
6.1	Darstellung des Beispielprozesses <i>LoanProcess</i> in einem BPEL-Editor	68
6.2	Darstellung des modifizierten Prozessausschnittes des <i>LoanProcess</i>	69

Verzeichnis der Codelistings

2.1	Vereinfachte Darstellung der Standardattribute und -elemente einer BPEL- <i>Activity</i>	13
2.2	Beispiel einer Übergangsbedingung (BPEL-Link mit <i>Transition condition</i> und <i>Join condition</i>)	15
3.1	Einfügen einer Aktivität in einem <flow> unter Anwendung der ursprüngli- chen <i>Transition condition</i> auf die neu eingefügte Aktivität	35
3.2	Einfügen einer Aktivität in einem <flow> unter Beibehaltung der ursprüngli- chen <i>Join condition</i> für die vorhande Aktivität B	36
3.3	Einfügen einer Aktivität unter Beibehaltung der <i>join condition</i>	39
3.4	Entfernen einer Aktivität aus einem <flow> unter Beibehaltung der ursprüng- lichen <i>Transition conditions</i>	41
5.1	Beispielinhalt eines Webservice-Aufrufs zum Hinzufügen einer <i>Invoke</i> -Aktivität	58
6.1	Beispiel der Webservice-Operation zum Hinzufügen einer <i>Sequence</i>	69

Einleitung

Geschäftsprozesse spielen in nahezu jedem Bereich der Wirtschaft eine tragende Rolle. Mit Hilfe eines Geschäftsprozesses wird die Abfolge einzelner Tätigkeiten beschrieben, die nötig sind um ein wirtschaftlich profitables Ziel zu erreichen.

Durch die zunehmende weltweite und unternehmensübergreifende Verknüpfung müssen Prozesse flexibel gestaltbar sein, um auf geänderte Bedingungen der Märkte, geänderte Produkte oder neue Strukturen bei Kunden oder Zulieferern schnell und effizient reagieren zu können. Die Durchführung von Geschäftsprozessen wird häufig durch Software unterstützt und die Softwareentwicklung in diesem Bereich spiegelt diesen Trend wider.

In der traditionellen Softwareentwicklung werden Geschäftsprozesse häufig in den Strukturen der Software nachgebildet. Bei einer Änderung des Prozesses ist deshalb oft ein tiefer Eingriff und eine Umstrukturierung des Programmcodes notwendig. Prozessänderungen können dadurch teuer und zeitintensiv werden, was zu starren Prozessen innerhalb von Unternehmen führen kann.

Im Unterschied dazu steht die Idee, den Geschäftsprozess von der Softwarefunktionalität zur Ausführung einzelner Tätigkeiten getrennt zu halten. Die Softwareunterstützung der einzelnen Prozessschritte wird dabei als Dienst bereitgestellt und kann entsprechend des Prozessverlaufes genutzt werden. Diese Vorgehensweise wurde in den letzten Jahren unter dem Begriff der *Service-orientierten Architektur (SOA)* bekannt und spielt eine immer stärkere Rolle in Unternehmen. Der Dienst wird dabei häufig in Form von *Web Services* zur Verfügung gestellt. Die entsprechende Methode des *Web Services* kann dann aus dem Prozess heraus genutzt werden. Prozesse werden bei diesem Vorgehen in einem Prozessmodell definiert. Der de facto Standard zur Modellierung eines solchen Prozessmodells ist die dafür entwickelte Sprache *BPEL (Business Process Execution Language)* [NLKL07].

Im Unterschied zur „traditionellen“ Softwareentwicklung bietet diese Verfahrensweise den Vorteil, dass die Funktionalität der Software bei einer Änderung des Prozessmodelles

unberührt bleibt. Benötigt ein Prozess zusätzliche Funktionalität, muss der bereitgestellte Dienst entsprechend erweitert werden. Dabei können andere Funktionen des Dienstes unberührt bleiben.

Die auf diese Art ausgeführten Prozesse können sowohl sehr kurzlebig sein als auch über einen sehr langen Zeitraum aktiv bleiben, wie zum Beispiel im Falle eines Kreditvertrages über viele Jahre.

1.1 Motivation und Ziel der Arbeit

In aller Regel hat ein Workflow-System den Zweck, einen Prozess nach dem vorgegebenen Prozessmodell auszuführen und den Benutzer auf die Einhaltung dieses Prozess einzuschränken. Durch die Trennung der Prozessdurchführung von der Ausführungsschicht wird zwar die Einführung eines neuen Prozessmodelles vereinfacht. Die Änderung eines bereits ablaufenden Prozessdurchlaufes ist dadurch jedoch nicht implizit möglich. Es können aber Fälle auftreten, in denen eine Modifikation des vorgegebenen Prozesses für einen aktiven Prozessdurchlauf nötig wird. Insbesondere bei langlebigen Prozessen kann es erforderlich werden, während der Ausführung in den Prozessverlauf einzugreifen, zum Beispiel wenn sich die gesetzliche Rahmenbedingungen ändern. Auch im Fehlerfall muss ggf. in den Prozessverlauf eingegriffen werden können [LR00, 3.7.3 Process Repair]. Die Durchführung einer Änderung an einem Prozessdurchlauf, kann als *Enforcement*¹ bezeichnet werden.

In dieser Arbeit werden die Schwierigkeiten bei der Änderung von Prozessinstanzen diskutiert und es werden Lösungsansätze zur Änderung laufender BPEL-Prozesse dargestellt. Mit der Arbeit wird auch ein Prototyp einer solchen Lösung für die Open Source Workflow Engine *Apache ODE* erstellt.

1.2 Aufbau der Arbeit

Kapitel 2 - Grundlagen In diesem Kapitel werden die Grundlagen von BPEL vorgestellt, sowie weitere wichtige Begriffe geklärt. Außerdem wird der aktuelle Stand der Entwicklung beschrieben.

Kapitel 3 - Workflowmodifikation Im dritten Kapitel werden die Gründe zur Workflowmodifikation, die Schwierigkeiten und die Änderungsoperationen behandelt.

Kapitel 4 - Konzepte Das Kapitel 4 enthält die Konzeption möglicher Lösungen der Instanzmodifikation und damit verbundene Schwierigkeiten.

¹*Enforcement*: Englisch für *Durchführung* oder *Erzwingung*

Kapitel 5 - Entwurf & Realisierung In diesem Kapitel findet sich der Entwurf der Realisierung des Konzeptes für das WMS *Apache ODE*.

Kapitel 6 - Anwendungsbeispiel Im sechsten Kapitel findet sich ein Anwendungsbeispiel der Instanzmodifikation.

Kapitel 7 - Fazit Abschließend werden die Ergebnisse der Arbeit zusammengefasst und es wird ein Ausblick auf die mögliche weitere Entwicklung der Prozessmodifikation gegeben.

1.3 Typographische und formale Konventionen

Definitionen bestimmter wiederkehrender Begriffe werden *kursiv* dargestellt.

Alle BPEL-bezogenen Ausführungen beziehen sich auf die Spezifikation *Web Services Business Process Execution Language Version 2.0* [OASo7].

Codebeispiele sind in Rahmen gefasst und verwenden eine nicht-proportionale Schriftart. BPEL-Codebeispiele können Elemente enthalten, die nicht der BPEL-Spezifikation entsprechen. Für BPEL-Aktivitäten wird ein Element `<activity>` verwendet, das als Platzhalter für die BPEL-Basisaktivitäten `<empty>`, `<invoke>`, `<receive>`, `<reply>`, `<assign>`, `<validate>`, `<throw>`, `<wait>`, `<extensionActivity>`, `<exit>` oder `<rethrow>` oder für die BPEL-Strukturaktivitäten wie `<sequence>`, `<if>`, `<while>`, `<repeatUntil>`, `<pick>`, `<flow>` oder `<forEach>` stehen kann. Im Fließtext werden aus Programmbeispielen entnommene Klassennamen oder BPEL-Elemente ebenfalls in einer nicht-proportionalen Schrift gesetzt.

Grundlagen

2.1 Geschäftsprozesse

Unter einem Geschäftsprozess wird im Allgemeinen ein Ablauf eines Wertschöpfungsprozesses verstanden. In jedem Unternehmen laufen solche Prozesse ab, ob ein Produkt gefertigt wird oder Dienstleistungen, Versicherungen oder Kredite angeboten werden. Der Geschäftsprozess regelt dabei das Zusammenspiel der beteiligten Personen, der Maschinen und des Arbeitsmaterials sowie den Ablauf der benötigten Arbeitsschritte. Ein Geschäftsprozess wird als administrativ bezeichnet, wenn mit dem Prozess interne Abläufe geregelt werden. Als operative Prozesse werden die Prozesse bezeichnet, in denen das Produkt entsteht [Hoh05]. Dabei hat sich in den letzten Jahren die Auffassung durchgesetzt, dass in vielen Fällen das Produkt dem Prozess entspricht. Soll das Produkt weiterentwickelt werden, muss sich auch der Prozess entwickeln. Mit einem verbesserten Prozess können Kosten eingespart werden, indem überflüssige Prozessschritte weggelassen werden oder durch die Zeitersparnis bei der Parallelisierung bestimmter Vorgänge.

Ein Geschäftsprozess kann auch Unternehmens- oder Standortübergreifend ablaufen, beispielsweise wenn Einzelteile eines Endproduktes von Zulieferern in anderen Teilen der Welt gefertigt werden oder sich das Callcenter eines deutschen Dienstleisters in Indien befindet. In solchen Fällen spielt die Zusammenarbeit über Staatsgrenzen und informationstechnisch verschiedene Systeme hinweg eine entscheidende Rolle.

2.2 Workflow-Modelle

Um einen Geschäftsprozess verständlich darstellen zu können, bedient man sich in aller Regel einfacher Ablaufdiagramme. Dabei werden die Abläufe mit Hilfe wiederkehrender Elemente

grafisch dargestellt. Die Beschreibung eines Geschäftsprozesses kann auch natürlichsprachlich oder mit Hilfe einer formalisierten Sprache geschehen. Jede dieser Beschreibungsarten stellt ein Prozessmodell dar, mit dessen Hilfe der beschriebene Prozess durchgeführt werden kann. Ein solcher Prozess kann vollkommen ohne Unterstützung von Softwaresystemen ablaufen, oder teilweise oder vollständig mit Hilfe von Software abgebildet und unterstützt werden. Zur Durchführung eines Geschäftsprozesses mit Hilfe von Software, muss der Prozess in einer von der Software zu verarbeitenden Sprache definiert werden. BPEL ist eine solche Sprache und wird im Abschnitt 2.4 beschrieben. Ein mit Hilfe von Software durchgeführtes Prozessmodell wird als *workflow model* bezeichnet [LRoo].

2.3 Workflow-Instanzen

Eine *Workflow-Instanz* ist ein konkreter Fall eines durch ein *Workflow-Modell* beschriebenen Prozesses. Eine *Workflow-Instanz* verfügt somit zu jedem Zeitpunkt über einen Status und im Prozess-Modell kann der jeweils aktuelle Zustand einer *Workflow-Instanz* festgestellt werden. Aus einem *Workflow-Modell* kann eine sehr große Anzahl von *Workflow-Instanzen* hervorgehen, es kann aber auch *Workflow-Modelle* geben, die lediglich einmal ausgeführt werden oder die nie zur Ausführung kommen. So wird beispielsweise der *Workflow* zur Produktion eines bestimmten Fahrzeug-Modells viele tausend Mal ausgeführt, während das *Workflow-Modell* zum Bau eines bestimmten Satelliten nur ein einziges Mal durchgeführt wird.

Der Zustand einer *Workflow-Instanz* setzt sich zusammen aus dem Inhalt der Prozessvariablen und dem Prozessfortschritt.

Auch der bisherige Verlauf der *Workflow-Instanz* ist Teil ihres Zustandes. Dieser Verlauf wird bei der Durchführung des Prozesses in der *Workflow engine* gespeichert und heißt *Audit trail* [LRoo].

2.4 BPEL

WS-BPEL (Web Services Business Process Execution Language) ist ein Sprachstandard zur Beschreibung von Geschäftsprozessen in einem XML-Dialekt. *BPEL* wurde im Jahr 2002 von IBM, BEA Systems und Microsoft eingeführt und gilt heute als de facto Standard zur *Workflow-Modellierung* [KHC⁺05]. Im Jahr 2007 wurde der aktuelle Standard *BPEL 2.0* von der Organisation OASIS verabschiedet [OASo7].

Mit *BPEL* steht ein Standard zur Verfügung, der es erlaubt *Webservices* zu *orchestrieren*. Das bedeutet, die Reihenfolge der Ausführung und die Verknüpfung von Webservice-Aufrufen kann definiert werden. Die Prozessschritte eines BPEL-Workflows werden in Form von Webservice-Aufrufen realisiert. Dadurch ist die systemübergreifende Nutzung verschiedener Dienste möglich. Auch der BPEL-Workflow selbst wird als Webservice zur Verfügung gestellt, was ein transparentes Einbinden eines Geschäftsprozesses erlaubt, ohne dass nach außen sichtbar wird, dass hinter dem Webservice-Aufruf ein ganzer Prozess steht. Dies erlaubt es, Prozesse auf gut verwaltbare Ebenen herunterzubrechen. Das *Orchestrieren* eines BPEL-Prozesses wird auch als „Programmieren im Großen“ bezeichnet, da dabei funktional eigenständige Blöcke verknüpft werden. Die Ausführungsschicht in der die Funktionalität der Prozesselemente umgesetzt wird, wird in diesem Zusammenhang im Gegensatz zur Prozesssteuerung als „Programmieren im Kleinen“ bezeichnet.

2.4.1 Struktur der Sprache

BPEL ist eine *XML (Extensible Markup Language)*-basierte Sprache [W3Co8] [OASo7]. Die Definition eines BPEL-Prozesses besteht aus zwei Teilen: Der Beschreibung der Webservice-Schnittstelle in einer *WSDL*-Datei und der Beschreibung des Prozesses in einer *BPEL*-Datei. Die Dateien enthalten Definitionen einer Reihe von Elementen. Zur Prozessbeschreibung in *BPEL* werden *Variablen*, *Correlations*, *Scopes*, *Links* und *Aktivitäten* angegeben. Zur Verknüpfung der *Aktivitäten* mit den Webservice-Operationen werden *Partner Link Types*, *Partner Links* und *Endpoints* angegeben. Diese Elemente werden im Folgenden näher erläutert.

Die Prozessdefinition beginnt in der ersten Ebene mit dem `<process>`-Tag. Darin enthalten sind die Definitionen der `<extensions>`, `<import>`, `<partnerLinks>`, `<messageExchanges>`, `<variables>`, `<correlationSets>`, `<faultHandlers>` und `<eventHandlers>`, sowie einer Hauptaktivität. Mit den in der Hauptaktivität enthaltenen weiteren Aktivitäten wird der Prozessverlauf festgelegt.

2.4.2 Scopes

Ein *Scope* stellt einen Behälter für die darin eingeschlossenen Aktivitäten dar. Für einen *Scope* können *Event Handler*, *Fault Handler* und *Compensation Handler* festgelegt werden, die innerhalb dieses Scopes gültig sind. Auch *Variablen*, *Partner Links* und *Correlation Sets* können für einen *Scope* definiert werden. Der gesamte BPEL-Prozess ist selbst ein *Scope* und verfügt damit über die selben Elemente.

2.4.3 Variablen

Variablen in BPEL enthalten die Daten der jeweiligen Prozessinstanz und sind damit ein Teil der Zustandsdefinition einer Instanz. Jede Variable gehört zu einem *Scope*. Auch der globale Prozess stellt einen *Scope* dar und dazu gehörende Variablen werden als globale Variablen bezeichnet. Eingeschlossene *Scopes* können selbst Variablen definieren, die nur innerhalb dieses *Scopes* gültig sind. Die Zuweisung von Werten an Variablen erfolgt in BPEL in der `<assign>`-Aktivität, die im Abschnitt 2.4.4 beschrieben wird.

2.4.4 Aktivitäten

Es gibt zwei Arten von BPEL-Aktivitäten: *Basic activities* (Basisaktivitäten) und *Structured activities* (Strukturaktivitäten). *Basic activities* beschreiben die elementaren Schritte des Prozessverhaltens. *Structured activities* beschreiben dagegen den Kontrollfluss des Prozesses und können selbst wiederum Aktivitäten beinhalten.

Alle BPEL-Aktivitäten können über Standardattribute und -elemente verfügen, die in Codelisting 2.1 dargestellt sind. Die Angabe dieser Elemente ist optional. Mit den Elementen `<targets>` und `<sources>` werden Abhängigkeiten zwischen Aktivitäten innerhalb einer `<flow>`-Aktivität definiert.

Codelisting 2.1 Vereinfachte Darstellung der Standardattribute und -elemente einer BPEL-Activity

```
<activity name="NCName" suppressJoinFailure="yes|no">
  <targets>
    <joinCondition>bool-expr</joinCondition>
    <target linkName="NCName" />
  </targets>

  <sources>
    <source linkName="NCName">
      <transitionCondition>bool-expr</transitionCondition>
    </source>
  </sources>
</activity>
```

Basic activities

Basic activities sind atomare Aktivitäten innerhalb eines *BPEL-Workflows*, d. h. diese Aktivitäten enthalten selbst keine weiteren Aktivitäten, sondern beschreiben die einzelnen Schritte des Prozessverhaltens. Zu den *Basic activities* zählen unter anderem `<invoke>`, `<receive>`, `<reply>`, `<assign>`, `<wait>` und `<empty>`. Die Aktivität `<invoke>` dient dazu, eine externe Webservice-Methode aufzurufen, während bei der `<receive>`-Aktivität auf den Eingang einer Nachricht für die Aktivität gewartet wird. Mit einer `<assign>`-Aktivität können Prozessvariablen Werte zugewiesen werden. Dies geschieht mit Hilfe der XML-Abfragesprache XPath, womit auch die Verwendung komplexerer Ausdrücke ermöglicht wird. Die `<wait>`-Aktivität wird verwendet, um im Prozessverlauf eine bestimmte Zeitdauer oder einen bestimmten Zeitpunkt abzuwarten. Die `<empty>`-Aktivität führt keine Aktion durch und kann zum Beispiel zur Synchronisation parallel verlaufender Prozesszweige verwendet werden.

Structured activities

Structured activities sind BPEL-Aktivitäten zur Steuerung des Kontrollflusses eines BPEL-Prozesses. Die *Structured activities* `<while>`, `<repeatUntil>` und `<forEach>` dienen der wiederholten Ausführung von Aktivitäten, die Konstrukte `<pick>` und `<if>` zur bedingten Ausführung der enthaltenen Aktivitäten.

Aktivitäten innerhalb eines `<sequence>`-Elements, werden sequentiell ausgeführt. Nach der erfolgreichen Ausführung einer Aktivität innerhalb der `<sequence>`, wird die im Prozessmodell an nächster Stelle stehende Aktivität gestartet.

Flow activity

Eine `<flow>`-Aktivität ist eine weitere *Structured activity*. In der `<flow>`-Aktivität können *Basic-* oder *Structured activities* enthalten sein. Im Unterschied zur *Sequence* werden Aktivitäten innerhalb eines `<flow>`-Elements parallel ausgeführt, sofern keine Kontrollabhängigkeiten zwischen den Aktivitäten festgelegt sind. Die enthaltenen Aktivitäten können jedoch über `<link>`-Elemente miteinander verknüpft werden, wenn Kontrollabhängigkeiten definiert werden sollen. Parallel ablaufende Aktivitäten können mit Hilfe von *Links* und einer *Join condition* wieder zusammengeführt werden. Der *Scope* des *Flows* wird verlassen, wenn alle enthaltenen Aktivitäten beendet wurden.

2.4.5 Link (Übergang)

Ein `<link>` dient in BPEL dazu, Kontrollabhängigkeiten zwischen Aktivitäten innerhalb einer `<flow>`-Aktivität festzulegen. Ein `<link>` befindet sich immer in einem der drei Status *unset*, *true* oder *false*. *Links* werden als Element der `<flow>`-Aktivität angegeben und müssen dann innerhalb dieses *Flows* als Beginn (*Source*) und Ziel (*Target*) jeweils einer enthaltenen Aktivität angegeben werden. Der Name eines *Links* muss innerhalb eines *Flows* eindeutig sein, und ein *Link* kann nur einmal als Ursprung und Ziel angegeben werden, so dass durch jeden *Link* zwei Aktivitäten verbunden werden.

2.4.6 Transition Condition (Übergangsbedingung)

Wie in Codelisting 2.2 zu sehen, kann der Beginn einer Kontrollabhängigkeit (*Link*) über eine *Transition condition* verfügen. Mit Hilfe der *Transition condition* wird bestimmt, ob ein *Link* vom Status *unset* in den Status *true* oder *false* wechselt. Die *Transition condition* wird nach dem erfolgreichen Ausführen der Aktivität ausgewertet, von der der *Link* ausgeht.

Codelisting 2.2 Beispiel einer Übergangsbedingung (BPEL-Link mit *Transition condition* und *Join condition*)

```
<flow>
  <links>
    <link name="AtoB"></link>
  </links>
  <invoke name="ActivityA">
    <sources>
      <source linkName="AtoB">
        <transitionCondition>$variableA</transitionCondition>
      </source>
    </sources>
  </invoke>
  <invoke name="ActivityB">
    <targets>
      <joinCondition>$AtoB</joinCondition>
      <target linkName="AtoB"></target>
    </targets>
  </invoke>
</flow>
```

2.4.7 Join Condition (Zusammenführungsbedingung)

Eine *Join condition* (Übergangsbedingung) kann für jede BPEL-Aktivität angegeben werden. Befindet sich eine Aktivität innerhalb eines `<flow>`, wird die *Join condition* über die eingehenden *Links* ausgewertet. Eine *Join condition* kann keine Prozessvariablen auswerten, sondern hat nur Zugriff auf den Status der eingehenden *Links*. Die zugehörige Aktivität wird ausgeführt, wenn der Ausdruck der *Join condition* zu *true* ausgewertet wird. Wird keine explizite *Join condition* angegeben, wird als Standard die Disjunktion der *Links* angenommen. Das bedeutet es genügt, wenn die *Transition condition* eines einzigen *Links* zutreffend ist, um die zusammenführende Aktivität auszuführen.

2.4.8 Dead-Path-Elimination

Um Situationen zu vermeiden in denen eine *Join condition* niemals erfüllt werden kann, wird die sogenannte *Dead path elimination* angewandt. Kann eine Aktivität aufgrund einer nicht erfüllten *Join condition* nicht gestartet werden, geht sie in den Status *Skipped (Dead)* über. Alle von der Aktivität ausgehenden Kontrollabhängigkeiten werden in diesem Fall auf *false* gesetzt. Dieses Verhalten wird fortgesetzt, bis der Pfad auf eine *Join condition* trifft, die entweder noch nicht auswertbar ist, oder die erfüllt wird.

In BPEL gibt es zudem die Eigenschaft `suppressJoinFailure` für jeden `<scope>`. Der Standardwert für diese Eigenschaft ist *no*. Das bedeutet, statt des beschriebenen Vorgehens der *Dead path elimination*, wird ein *Fault* geworfen, der wiederum von einem *Fault handler* abgefangen werden kann. Hat die Eigenschaft `suppressJoinFailure` jedoch den Wert *yes*, wird die *Dead path elimination* wie beschrieben angewandt.

2.5 Workflow Engines

Eine *Workflow engine* ist Software zur Verwaltung und zur Ausführung von *Workflow-Prozessen*. Die *Workflow engine* sorgt dafür, dass Prozessinstanzen gestartet und die Aktivitäten ausgeführt werden. Zudem sorgt eine *Workflow engine* für die Persistenz des Instanzstatus.

Es gibt *Workflow engines* die jeweils unterschiedliche Sprachen zur Definition von *Workflows* verwenden können. BPEL wird von einigen modernen *Workflow engines* unterstützt, unter anderem von *Apache ODE*.

2.6 Apache ODE

Apache ODE (Orchestration Director Engine) [Apa] ist eine Open-source *Workflow engine*, die aus einer kommerziellen Entwicklung hervorging. *Apache ODE* unterstützt den Sprachstandard WS-BPEL 2.0 und die vorhergegangene Spezifikation BPEL4WS 1.1. Die *Workflow engine* nutzt zur Webservice-Kommunikation das *Apache Axis2*-Projekt [Apa09]. *Apache ODE* unterstützt über die BPEL-Spezifikation hinaus einige Erweiterungen des Standards, darunter die Nutzung der Abfragesprache *XPath* in der Version 2.0 und die Verwendung externer Variablen, die bei Verwendung im Prozess aus einer externen Datenbank gelesen und bei Änderung des Wertes dort gespeichert werden können.

Instanzmodifikationen sind in *Apache ODE* bisher nicht möglich. Wird eine aktualisierte Version eines Prozesses mit dem selben Namen deployed, werden alle Instanzen des Prozesses gelöscht. Für die Praxis bedeutet dies, dass geänderte Prozesse unter einem neuen Namen veröffentlicht werden müssen, wenn noch Instanzen des Prozessmodells aktiv sind, die nicht verloren gehen sollen.

2.6.1 Management API

Für *Apache ODE* steht eine *Management API* als *Webservice* zur Verfügung. Diese API besteht aus den Schnittstellen *ProcessManagement* und *InstanceManagement*.

Process Management

Die Schnittstelle *ProcessManagement* dient der Verwaltung der verfügbaren Prozesse. Mit dieser Schnittstelle können verfügbare Prozesse aufgelistet, aktiviert oder deaktiviert werden. Mit Hilfe der Methode `listAllProcesses()` können die verfügbaren Prozesse aufgelistet werden. Die dabei gelieferten Informationen enthalten unter anderem die Prozess-ID, den Zustand, die Version, den Status und die Endpunkte der Prozesse. Diese Informationen können für einen einzelnen Prozess auch über die Methode `getProcessInfo()` abgerufen werden.

Instance Management

Mit der Schnittstelle *InstanceManagement* können die Prozessinstanzen der *Workflow engine* verwaltet werden. Über die Methode `listInstances()` können alle Instanzen aufgelistet werden. Dabei ist es auch möglich, die Instanzen auf bestimmte Eigenschaften zu filtern und beispielsweise nur Instanzen eines bestimmten Prozesses anzuzeigen oder Informationen

über eine Instanz mit einer bestimmten *Correlation property* zu erhalten. Zudem kann mit Hilfe der Methode `getInstanceInfo()` auch gezielt die Information einer bestimmten Instanz unter Angabe der Instanz-ID abgerufen werden.

Die gelieferten Informationen enthalten auch die Angabe der Scope-ID des Prozesses. Die Methode `getScopeInfoWithActivity()` liefert wiederum Informationen über den Zustand eines *Scopes* und der darin enthaltenen *Variablen* und *Scopes*. Diese Informationen enthalten auch die IDs der *Variablen* und *Scopes*.

Der Webservice *InstanceManagement* erlaubt es zudem, Instanzen zu beenden, zu löschen, anzuhalten und fortzusetzen. Eine Modifikation der Instanz wie sie in dieser Arbeit beschrieben wird, ist über den Service jedoch nicht möglich. Die über diesen Service abgerufenen Informationen über den Zustand der Prozessinstanzen können aber für die Modifikation verwendet werden, um Instanzen und Aktivitäten zu identifizieren.

2.7 Beispielprozesse

Ein einfaches Beispiel für einen BPEL-Prozess findet sich in der BPEL-Spezifikation [OASo7]. Der Prozess „PurchaseOrderProcess“ stellt einen Bestellvorgang modelliert in BPEL dar und wird in Abbildung 2.1 dargestellt.

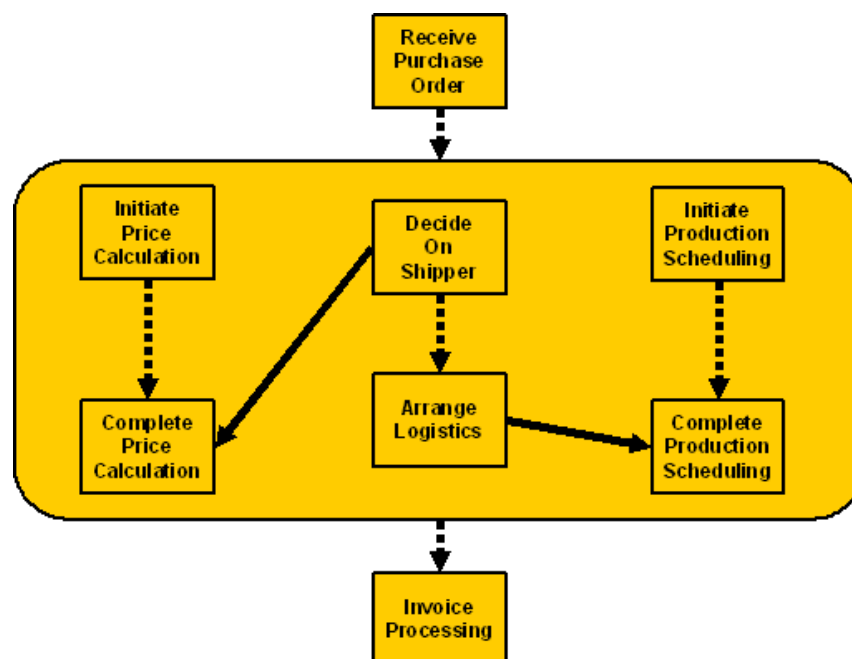


Abbildung 2.1: Bestellprozess (Purchase Order Process) [OASo7]

Beim Empfang der Bestellung werden drei Aktivitäten parallel gestartet: Die Berechnung des Preises, die Zuordnung des Logistikers und der Beginn der Bereitstellung. Teile des Prozesses können parallel ausgeführt werden, es gibt jedoch auch Abhängigkeiten zwischen den Aktivitäten. Zur vollständigen Preisberechnung werden die Versandkosten benötigt, und zur Bereitstellung der Bestellung ist das Versanddatum nötig. Diese Datenabhängigkeiten sind im Diagramm als durchgehender Pfeil dargestellt, während die sequentielle Abfolge der Aktivitäten durch gestrichelte Pfeile dargestellt ist.

Eine denkbare Änderung an diesem Beispielprozess wäre, den Kunden zu informieren, wenn der Versandtermin der Bestellung feststeht. Dazu muss eine neue Aktivität „Advise Customer“ eingefügt werden. Diese neue Aktivität soll nach „Complete Production Scheduling“ ausgeführt werden und den Kunden über den Liefertermin und den Spediteur in Kenntnis setzen. Natürlich wäre es möglich, diesen Prozessschritt bereits beim Erstellen des Prozessmodells zu berücksichtigen. In diesem Beispiel wird jedoch davon ausgegangen, dass diese Funktion neu in das System eingeführt werden soll und auch bei laufenden Bestellungen zum Einsatz kommen soll.

Ein weiteres Beispiel für einen BPEL-Prozess ist der in Abbildung 2.2 dargestellte Kreditantrag. Dieses Beispiel wird in dieser Arbeit zur Darstellung der Workflowmodifikation verwendet und der BPEL-Code und der WSDL-Code sind im Anhang A zu finden.

Mit dem Aufruf der Aktivität „Take out loan“ wird eine neue Instanz des Geschäftsprozesses gestartet. Beim Aufruf der Webservice-Methode werden die Daten des Kreditnehmers, die Höhe des Kredites und eine Vorgangsnummer an die *Workflow engine* übergeben. In der *Workflow engine* werden diese Daten gespeichert und die neue Prozessinstanz betritt eine Schleife, in der mit in der Aktivität „Request status“ der Status des Kreditkontos abgefragt werden kann, oder in der Aktivität „Pay installment“ eine Rate bezahlt werden kann. Diese Schleife wird wiederholt, bis der Kredit abbezahlt wurde. Anschließend wird in einer letzten Aktivität „Finalize loan“ der Kreditvertrag beendet.

Dieses Beispiel eines Kreditprozesses ist zwar stark vereinfacht, aber ein ähnlicher Prozess wäre für den Anwendungsfall vorstellbar. Die Laufzeit eines solchen Prozesses beträgt oft mehrere Jahre. Wir nehmen als Beispiel an, dass es eine Reihe von Prozessinstanzen dieses Prozesses gibt und dass aufgrund einer neuen Vorschrift diese laufenden Prozessinstanzen geändert werden müssen. Eine zentrale europäische Einrichtung verlange die Meldung der Kreditsumme jedes offenen Kreditvertrages an einen bereitgestellten Webservice. In diesem Beispiel müsste also eine weitere Aktivität „Update supervision“ eingefügt werden, die jedesmal ausgeführt wird, nachdem die Aktivität „Pay installment“ durchgeführt wurde. Wie eine solche Änderung durchgeführt werden kann, welche Probleme dabei auftreten und wie diese Probleme behandelt werden können, wird in den folgenden Kapiteln behandelt.

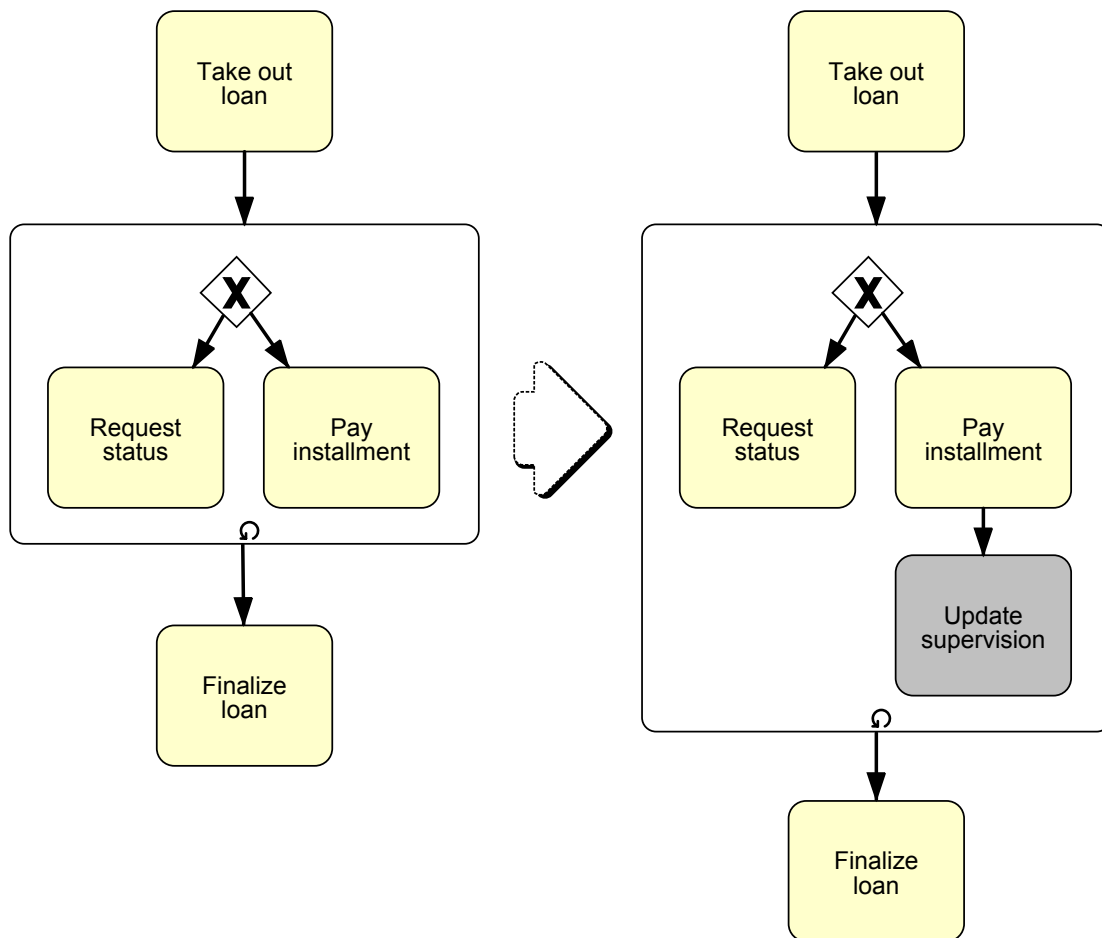


Abbildung 2.2: Vereinfachtes Beispiel eines Kreditprozesses vor und nach dem Hinzufügen einer Aktivität

Workflowmodifikation

Ein Workflowmodell wird normalerweise erstellt, um einen Geschäftsprozess einheitlich steuern zu können und möglichst effizient mit gegebenen Ressourcen wie Mitarbeitern, Maschinen, Arbeitsmaterial und Arbeitsplätzen umzugehen. Der Verlauf des Prozesses ist im Prozessmodell weitgehend optimiert und sollte nicht verändert werden.

Trotzdem kann es eine Reihe von Gründen geben, die es nötig machen den Prozessverlauf zu modifizieren [AJo0]. Zu diesen Gründen kann die Einführung neuer Produkte oder Servicemodelle zählen, beispielsweise wenn dafür zusätzliche Daten benötigt werden. Auch Änderungen an gesetzlichen Vorgaben oder an der Hardwareausstattung können dazu führen, dass in einen Prozess eingegriffen werden muss.

Neben diesen Gründen, deren Ursachen in der Umgebung des Workflowsystems liegen, kann es auch Ursachen im Workflowsystem selbst geben. Dazu zählen beispielsweise Fehler in der Prozessdefinition, die erst im Verlauf des Prozesses auffallen oder etwa technische Probleme. Denkbar ist auch eine nachträgliche Verbesserung eines Prozesses, beispielsweise durch Parallelisierung entsprechender Aktivitäten. Eine große Rolle bei der Workflowmodifikation spielen auch Ausnahmefälle, die unter Umständen nicht alle bei der Modellierung des Prozesses berücksichtigt werden können oder übersehen werden.

Die Durchführungszeit einer Prozessinstanz kann zwischen Sekunden und Jahren liegen. So wäre ein Prozess eines Kreditverfahrens vorstellbar, bei dem der Kreditnehmer seine Schulden über mehrere Jahre abbezahlt. Insbesondere bei solch langlaufenden Prozessen sind Änderungen leicht vorstellbar. Für das im Kapitel 2.7 bereits erwähnte Beispiel eines Kreditverfahrens könnte eine neue gesetzliche Regelung eingeführt werden, die die Bank zu einer regelmäßigen Meldung des Kreditstandes an eine zentrale Stelle verpflichtet. Diese Änderung muss auch für bestehende Kreditverfahren übernommen werden. Aber auch bei kurzlebigen Prozessen können Sonderfälle auftreten, die einen Eingriff in die Prozessinstanz nötig machen.

Eine Prozessmodifikation kann grundsätzlich auf zwei Ebenen stattfinden – auf Ebene des Prozessmodells oder auf Ebene der konkreten Prozess-Instanz. Auf Ebene des Prozessmodells erfolgen Änderungen, die für alle Prozessinstanzen gültig sein sollen, also beispielsweise eine Optimierung des Prozessverlaufs. Dabei wird von *Schema evolution* gesprochen. Im Unterschied dazu, wird das Ändern einzelner Prozessinstanzen als *Instanzmodifikation* (*Instance modification*, *Instance adaptation*) bezeichnet [WRMR07].

3.1 Workflow-Schema-Änderungen

Die Prozessmodifikation findet auf Ebene des Prozessmodells statt, wenn Fehler im Prozessmodell gefunden werden, wenn das Prozessmodell optimiert werden soll, oder wenn sich die Umgebung geändert hat. Bei der Änderung eines Prozessmodells stellt sich die Frage, auf welche der Prozessinstanzen die Änderung angewandt werden soll. Müssen alle laufenden Prozessinstanzen geändert werden, oder genügt es einige ausgewählte Instanzen zu ändern? Diese Entscheidung ist hauptsächlich abhängig von der Ursache, die die Änderung erforderlich macht.

Es gibt mehrere Möglichkeiten, wie bei einer Änderung auf Ebene des Workflow-Modells mit aktiven Prozessinstanzen verfahren werden kann. In [SO99] und [AJ00] beschreiben die Autoren vier Verfahren zum Umgang mit Prozessinstanzen.

Flush Die einfachste Methode, mit Änderungen am Prozessmodell zu verfahren, ist das sogenannte *Flush* [SO99] bzw. *Proceed* [AJ00]. Dabei laufen bereits gestartete Instanzen nach dem alten Prozessmodell weiter. Werden neue Prozessinstanzen gestartet, übernehmen diese das neue Prozessmodell. Die Anwendung des *Flush*-Verfahrens ist also nur für Prozessänderungen geeignet, bei denen die laufenden Instanzen nach dem alten Prozessmodell weiterlaufen können.

Abort Bei Anwendung von *Abort* [SO99] bzw. *Forward recovery* [AJ00] wird die Ausführung laufender Instanzen des zu ändernden Prozesses unabhängig von ihrem Zustand abgebrochen. Die Instanzen können dann unter Anwendung des neuen Prozessmodells erneut gestartet werden. Diese Methode ist natürlich nur praktikabel, wenn der Abbruch eines laufenden Prozesses keine massiven Mehrkosten im Vergleich zur Sonderbehandlung dieser Vorgänge verursacht.

Migrate Wird bei der Änderung eines Prozessmodells versucht, alle laufenden Instanzen auf die aktualisierte Version des Prozessmodells zu übernehmen, spricht man von *Migrate* [SO99] bzw. *Transfer* [AJ00]. Dabei ist es im Allgemeinen nicht möglich, tatsächlich alle Prozessinstanzen zu migrieren [Ley09]. Es können lediglich die Instanzen migriert werden, bei denen die Ausführung sämtlicher Aktivitäten, auf die sich die Änderung auswirkt, zum Zeitpunkt der Änderung in der Zukunft liegt.

Adapt Bei Anwendung von *Adapt* [SO99] werden lediglich die Instanzen eines Prozesses geändert, die eine Ausnahme darstellen und eine vom Prozessmodell abweichende Behandlung benötigen. Das bedeutet, im Unterschied zu *Migrate*, wird nicht versucht alle Instanzen auf das neue Prozessmodell zu migrieren. Stattdessen bestimmt der *Workflow administrator (WEA)* die Instanzen für die eine Migration notwendig ist.

Bei Anwendung der Methoden *Migrate* oder *Adapt* werden laufende Prozessinstanzen geändert, was im folgenden Abschnitt beschrieben wird.

3.2 Workflow-Instanz-Änderungen

Im Unterschied zu Änderungen auf Ebene des Prozessmodells, können Änderungen auf Ebene der konkreten Prozessinstanz notwendig werden, wenn beispielsweise ein Ausnahmefall auftritt. Das Prozessmodell soll bei solchen Änderungen unverändert bleiben, die Änderungen sollen nur auf ausgewählte Prozessinstanzen angewandt werden.

Unabhängig davon ob einzelne Prozessinstanzen geändert, oder ob eine Menge von Instanzen auf Grund einer Schemaänderung modifiziert werden, müssen dabei eine Reihe von Bedingungen beachtet werden. Bei der Änderung einer geringen Anzahl oder einzelner Instanzen ist allerdings die individuelle Betrachtung jeder einzelnen Instanz durch den *Workflow administrator (WEA)* möglich. Dabei ist für den *WEA* ersichtlich, ob die gewünschte Modifikation möglich ist und es müssen nicht sämtliche Sonderfälle für die Änderung eines Prozessmodells beachtet werden.

Bei der Änderung gestarteter Prozesse gibt es eine Reihe von Problemen die gelöst werden müssen. Zunächst spielt der Zustand der betroffenen Instanz eine wichtige Rolle. Die Instanz kann nur geändert werden, wenn die betroffene Aktivität nicht bereits ausgeführt wurde. Die Details dieser Voraussetzungen werden in den folgenden Abschnitten behandelt.

3.2.1 Zustand einer Prozessinstanz

Ein Prozess durchläuft während seiner Ausführung verschiedene Status [LRoo]. Diese sind in Abbildung 3.1 dargestellt.

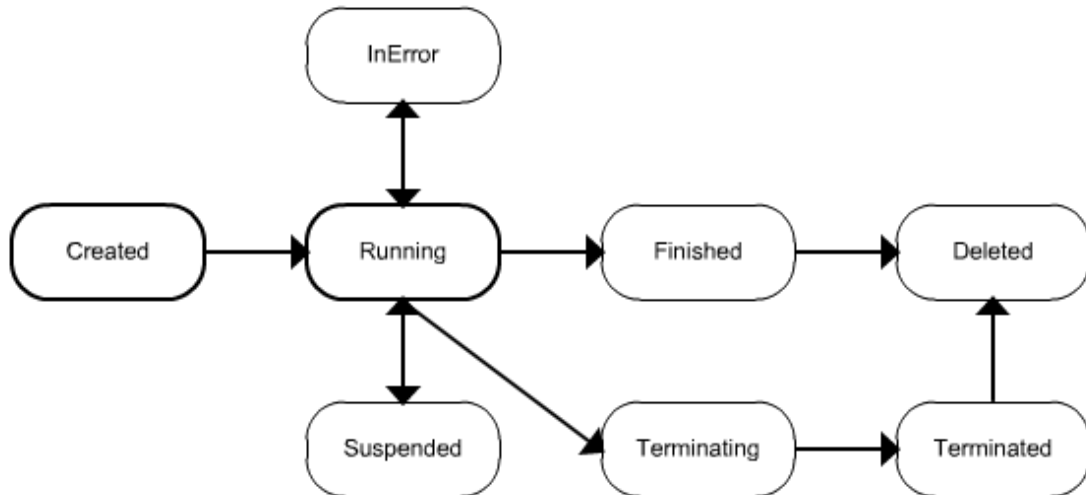


Abbildung 3.1: Zustände einer Prozessinstanz [LRoo]

Zunächst wird der Prozess aus dem Prozessmodell erstellt und befindet sich im Status *Created*. Mit der Ausführung der ersten Aktivität, wechselt der Prozess in den Status *Running*. So lange Aktivitäten ausgeführt werden, verbleibt der Prozess in diesem Status. Der Zeitraum zwischen der Ausführung von Aktivitäten kann sehr groß sein und der Prozess bleibt auch im Status *Running*, wenn Aktivitäten auf ein Ereignis warten.

Bei einem regulären Prozessverlauf bei dem keine unbehandelten Ausnahmefälle auftreten, erreicht der Prozess nach dem Abarbeiten der letzten Aktivität den Status *Finished*. In diesem Status bleibt der Prozess, es sei denn er wird durch einen Benutzereingriff oder zeitgesteuert nach Ablauf einer festgelegten Zeit vom Workflow-Management-System gelöscht.

Durch einen manuellen Eingriff kann der Prozess auch aus dem Status *Running* in den Status *Suspended* versetzt werden. In diesem Status wird nicht durch den Prozess navigiert und es werden keine Aktivitäten ausgeführt. Der Status *Suspended* wird erst wieder verlassen, nachdem dies durch einen weiteren Benutzereingriff oder zeitgesteuert veranlasst wurde. Im Fehlerfall während der Ausführung der Navigation oder einer Aktivität kann der Prozess in den Zustand *InError* übergehen. Ähnlich wie im Status *Suspended*, wird der Prozess dabei angehalten und der Workflow-Administrator kann die nötigen Schritte zur Wiederherstellung eines gültigen Zustandes der Prozessinstanz unternehmen.

Der Workflow-Administrator hat auch die Möglichkeit, einen Prozess manuell zu beenden. Dabei geht der Prozess in den Zustand *Terminating* und nach erfolgtem Beenden in den Zustand *Terminated* über.

Eine Instanzmodifikation kann für Prozessinstanzen sinnvoll sein, die sich in einem der Status *Running*, *Suspended* oder *InError* befinden. Der Status *InError* erfordert in aller Regel einen manuellen Eingriff, entweder durch eine Prozessmodifikation oder durch das Ändern der Inhalte von Variablen.

3.2.2 Zustand einer Aktivität

Der Zustand einer Aktivität spielt bei der Instanzmodifikation eine zentrale Rolle, da nur Aktivitäten geändert werden können, die noch nicht ausgeführt wurden. Abbildung 3.2 zeigt die Zustände einer Aktivität nach [LRoo].

Beim Erzeugen einer neuen Prozessinstanz werden alle Aktivitäten zunächst mit dem Zustand *Initial* angelegt. Start-Aktivitäten beginnen im Status *Executable*. Start-Aktivitäten sind Aktivitäten, die eine neue Instanz erzeugen können. Die eingehenden Kontrollabhängigkeiten (*Links*) einer Aktivität werden ausgewertet, wenn die vorhergehende Aktivität beendet wurde. Wenn die Auswertung der *Join condition false* ergibt, wechselt die Aktivität vom Zustand *Initial* in den Zustand *Dead*. Ergibt die Auswertung *true*, wechselt die Aktivität vom Zustand *Initial* in den Zustand *Executable* und wartet auf ihre Ausführung. Die Ausführung der Aktivität und der Statuswechsel vom Status *Executable* in den Status *Activated* beginnt im Falle von BPEL durch den Aufruf der zugehörigen Webservice-Methode, mit dem Erreichen eines definierten Zeitpunktes oder nach Ablauf einer bestimmten Zeit. In diesem Zustand wird die mit der Aktivität verbundene Aktion durchgeführt. Nach erfolgreicher Ausführung wechselt die Aktivität in den Status *Completed*. Kann die Aktivität nicht erfolgreich beendet werden, wechselt sie in den Zustand *Terminated*. Aus diesem Zustand kann die Aktivität wieder aktiviert werden, oder in den Zustand *Completed* übergehen, wobei dabei zwischen erfolgreicher und nicht erfolgreicher Ausführung unterschieden werden kann.

3.2.3 *Modification trace* und *Execution trace*

Aktivitäten im Prozessverlauf die im Kontrollfluss von einer modifizierten Aktivität abhängig sind, werden indirekt durch diese Modifikation beeinflusst. D. h. die Werte der Variablen können anders sein als vor der Änderung und Bedingungen z. B. in `<if>-<else>`-Verzweigungen können dadurch anders ausgewertet werden. Alle Aktivitäten für die dies zutrifft, können in einem *Modification trace* zusammenfasst werden. Keine dieser Aktivitäten darf vor der Änderung ausgeführt worden sein. In Abbildung 3.3 wird diese Menge grau hinterlegt

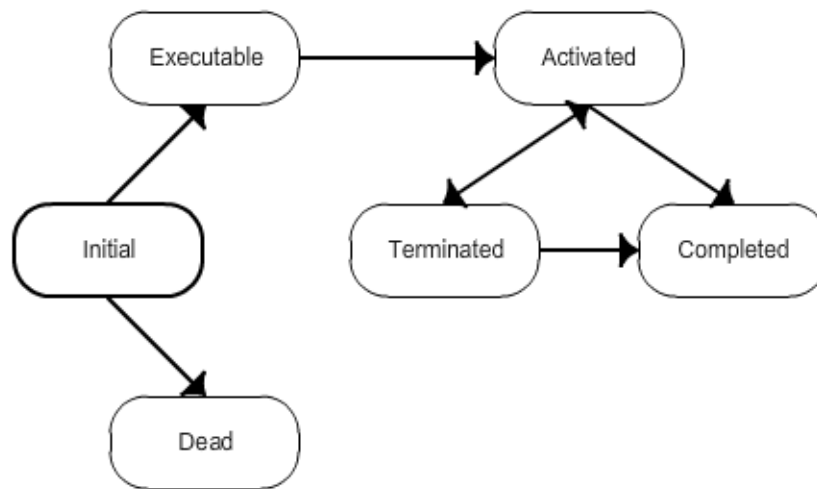


Abbildung 3.2: Zustände und Zustandsübergänge einer Aktivität [LRoo]

dargestellt. Modifizierte Aktivitäten sind mit *Änderung* markiert. Alle davon abhängigen Aktivitäten sind Teil des *Modification trace*.

Der Ausführungspfad (*Execution trace*) einer Prozessinstanz enthält alle Aktivitäten, die bereits gestartet wurden [WRMR07]. Für die Modifikation von Instanzen spielt der *Execution trace* eine wichtige Rolle bei der Entscheidung darüber, ob eine bestimmte Änderung für eine Prozessinstanz durchgeführt werden kann. Aktivitäten in einem Prozess, die zum Zeitpunkt der Modifikation bereits ausgeführt wurden, dürfen nicht von modifizierten Aktivitäten abhängig sein. In Abbildung 3.3 ist der *Execution trace* mit einer schwarzen Strichlinie umrandet. Die Menge der ausgeführten Aktivitäten darf sich nicht mit der Menge der Aktivitäten im *Modification trace* überschneiden.

Auf Prozessinstanzen, deren *Execution trace* sich mit dem *Modification trace* der Änderung überschneidet, kann die vorgesehene Änderung nicht angewendet werden [AJ00]. Eine unzulässige Änderung ist im Fall *d)* abgebildet. Es soll die selbe Modifikation wie in *b)* angewendet werden, der Verlauf der Prozessinstanz ist jedoch bereits weiter vorangeschritten. Somit würde sich bei der Anwendung dieser Änderung der *Modification trace* mit dem *Execution trace* überschneiden, das bedeutet die Änderung kann nicht durchgeführt werden.

3.2.4 Modifikation in Schleifen

Einen Sonderfall stellen Aktivitäten in Schleifenkonstrukten dar. Eine Schleife kann bereits mehrfach ausgeführt worden sein, bevor die Modifikation auf die Prozessinstanz angewandt wird. Der Änderungspfad umfasst in diesem Fall nicht die bereits ausgeführten Instanzen

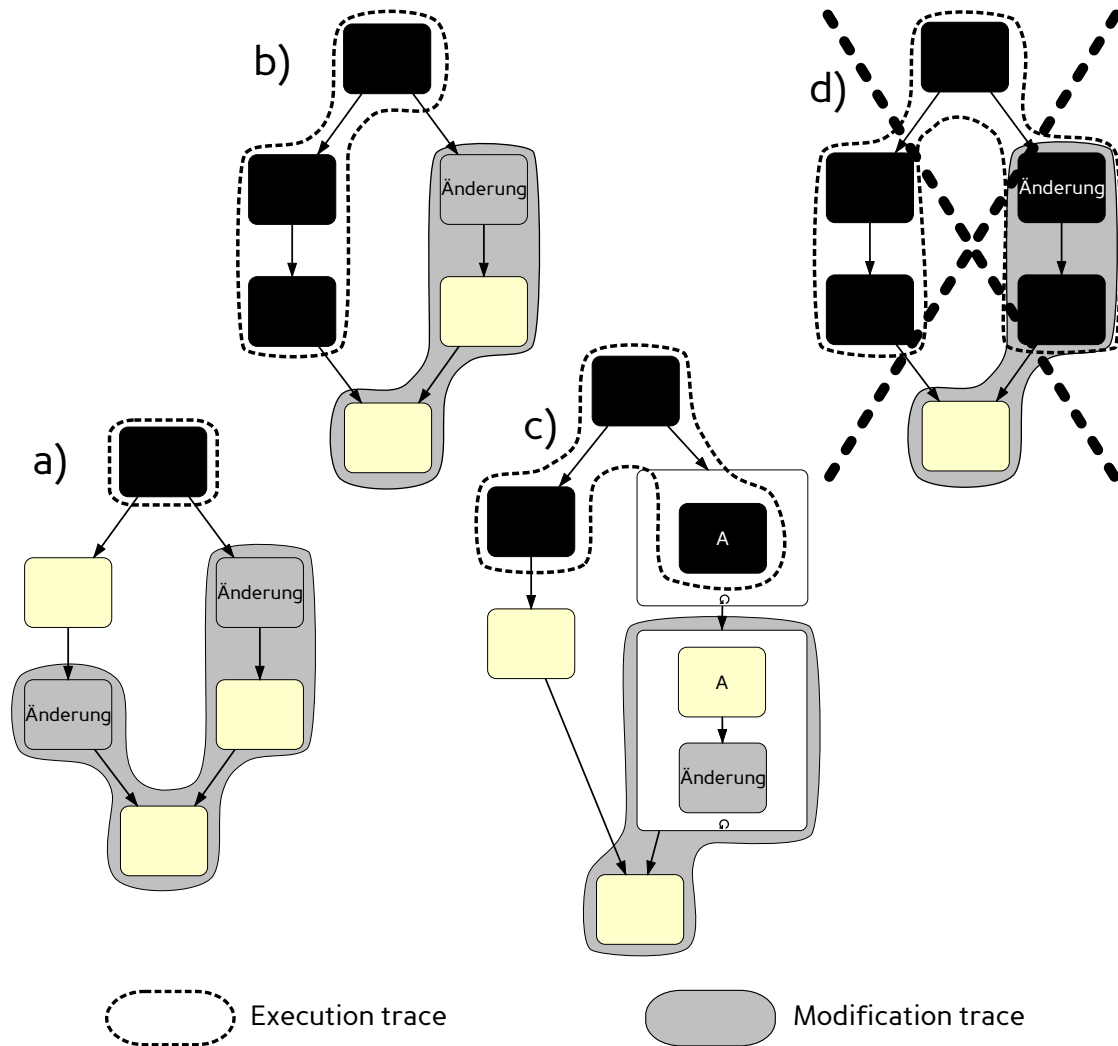


Abbildung 3.3: Execution trace und Modification trace

der Aktivität, sondern nur die Instanzen der Aktivität, die nach der Modifikation ausgeführt werden. Dieser Fall ist in Abbildung 3.3 im Fall c) dargestellt. Das bedeutet, dass eine solche Modifikation innerhalb einer Schleife möglich ist. Voraussetzung dafür ist, dass bei der Anwendung der Modifikation auf eine Instanz zwischen bereits durchgeführten Schleifendurchläufen und zukünftigen Schleifendurchläufen unterschieden werden kann. Dies ist vom Verfahren zur Durchführung der Modifikation abhängig.

3.3 Gültigkeit der Modifikation

3.3.1 Gültigkeitsbereiche von Variablen

Variablen werden in BPEL in einem *Scope* deklariert und „gehören“ zu diesem *Scope*. Auf Variablen eines *Scopes* kann innerhalb des *Scopes* und aus darin enthaltenen *Scopes* zugegriffen werden. Variablen mit dem selben Namen, die in einem inneren *Scope* deklariert werden, überlagern die Variablen des umschließenden *Scopes*. Da der BPEL-Prozess selbst ein *Scope* ist, können darin Variablen deklariert werden. Solche prozessweit gültigen Variablen heißen globale Variablen.

Im Zusammenhang mit der Modifikation von Prozessen, müssen die Gültigkeitsbereiche der Variablen beachtet werden. Zum Beispiel darf eine Aktivität die verschoben wird, nicht den *Scope* der Variablen verlassen, auf die in der Aktivität zugegriffen wird. Natürlich spielt auch die Reihenfolge des Datenzugriffs eine wichtige Rolle und wird in den folgenden Abschnitten weiter behandelt.

3.3.2 Abhängigkeiten im Kontrollfluss

Da durch die *Links* innerhalb eines *Flows* keine Schleifen gebildet werden können, wird Aktivität B immer ein Nachfolger der Aktivität A sein. *Links* dürfen in BPEL nicht über Grenzen von Schleifenkonstrukten hinweg gesetzt werden. Die *Transition Condition* und die *Join condition* sind deshalb transitiv auch für Aktivität B gültig, wenn Aktivität X eingefügt wurde. Dies setzt voraus, dass die entsprechenden Daten nicht durch Aktivität X ungültig werden.

$$ActivityA \longrightarrow ActivityX \wedge ActivityX \longrightarrow ActivityB \implies ActivityA \longrightarrow ActivityB$$

Eine Aktivität kann Ziel mehrerer *Links* sein. Soll eine solche Aktivität entfernt werden, muss dafür gesorgt werden, dass die eingehenden *Links* auf die nachfolgende Aktivität umgesetzt werden.

Im Falle einer <assign>-Aktivität kann es sein, dass durch das Entfernen der Aktivität eine Abbruchbedingung einer Schleife nie erfüllt wird.

Die zu ändernde Aktivität kann sich auch innerhalb einer Schleife befinden, so dass alleine aus der bereits erfolgten Ausführung nicht geschlossen werden kann, dass die Aktivität nicht erneut ausgeführt werden kann.

Das Entfernen einer Aktivität kann natürlich zu Problemen im Prozessverlauf führen. Wird beispielsweise eine `<assign>`-Aktivität entfernt, fehlt unter Umständen nachfolgenden Aktivitäten die Initialisierung oder die Zuweisung der Variablen.

3.3.3 Abhängigkeiten im Datenfluss

Eine *Datenabhängigkeit* zwischen zwei Aktivitäten besteht, wenn die Ausführung der zweiten Aktivität durch das Entfernen der ersten Aktivität beeinflusst wird. Wenn die Aktivität im Ursprung der *Datenabhängigkeit* entfernt wird, kann dies entweder das Ergebnis der Ausführung verändern, oder zu einem Fehler führen.

In BPEL könnte z. B. eine `<reply>`-Aktivität von einer `<assign>`-Aktivität abhängen, in der die Nachrichtenvariable initialisiert wird. Wird diese `<assign>`-Aktivität entfernt, führt dies zu einem `uninitializedVariable-Fault` im BPEL-Prozess.

Wenn die Variable jedoch bereits vor der `<assign>`-Aktivität initialisiert wurde, und in dieser nur verändert wird, führt das Entfernen der `<assign>`-Aktivität nicht zu einem Fehler in der Prozessausführung. Stattdessen hat die Änderung Auswirkung auf den Inhalt der Nachrichten der `<reply>`-Aktivität.

Wenn dagegen die Bedingung einer `<if>`-Aktivität von Daten einer `<assign>`-Aktivität abhängig ist, die eine bereits initialisierte Variable verändert, kann dies zu einem anderen Prozessverlauf führen.

Das Entfernen einer Aktivität von der eine *Datenabhängigkeit* ausgeht, kann also unterschiedliche Auswirkungen haben. Beim Anwenden einer Änderung auf eine laufende Prozessinstanz müssen diese *Datenabhängigkeiten* vom *Workflow administrator* berücksichtigt werden. Die Validierung der Änderungsoperation auf mögliche Verletzung der Abhängigkeiten geht über das Thema dieser Arbeit hinaus, ist aber Gegenstand weiterer Arbeiten, wie in Abschnitt 3.3.6 vorgestellt.

3.3.4 Überprüfung der Änderung

Beim Veröffentlichen eines BPEL-Prozesses in einer *Workflow engine*, wird der Prozess von der *Engine* auf Gültigkeit überprüft. Die selbe Überprüfung kann auch zur Validierung eines geänderten Workflow-Modells angewandt werden. Dazu wird die gewünschte Änderung in die Prozessdefinition des ursprünglichen Prozesses übernommen und der resultierende Prozess wird überprüft. Diese Art der Prüfung erlaubt jedoch nur die Überprüfung statisch feststellbarer Fehler. D. h. es können damit grundsätzliche Fehler einer Prozessänderung, z. B. fehlende Variablen oder ungültige Aktivitäts-Definition, erkannt werden. Nicht erkannt

werden können dabei Fehler, die zur Laufzeit der Prozessinstanzen auftreten und durch die Änderung ausgelöst werden. Dies könnte beispielsweise ein ungültiger Wert einer Variablen sein, oder Fehler in der Semantik des Prozesses.

3.3.5 Prozesssemantik

Die *Semantik* eines Prozesses bezeichnet die hinter einem Prozessmodell stehende Bedeutung der Ausführung und Ausführungsreihenfolge. Die Korrektheit der *Semantik* kann in einem BPEL-Prozess nach aktuellem Standard nicht überprüft werden. Eine semantische Bedingung in einem medizinischen Prozess könnte zum Beispiel sein, dass die Aktivität „Blutkonserven bereitstellen“ zwingend vor der Aktivität „Operation durchführen“ durchgeführt werden muss. Diese Bedingung muss in einem Prozess erfüllt bleiben, auch wenn die Aktionen im statischen Prozessmodell vertauscht werden können und das BPEL-Modell dadurch syntaktisch korrekt bleibt. Die Prozesssemantik muss also nicht nur bei der Definition eingehalten werden, sondern auch nach dem Durchführen von Änderungen. Da BPEL bisher keine Konstrukte zur Definition der Prozesssemantik bereitstellt, wird die Überprüfung in dieser Arbeit nicht weiter verfolgt und muss bei der Änderung eines Prozesses vom *Workflow administrator* vorgenommen werden.

3.3.6 Verwandte Arbeiten

In [LRDo8] wird ein Regelwerk beschrieben, das es erlaubt, modifizierte Prozesse auf ihre Gültigkeit zu validieren. Dabei werden Aktivitäten um Prädikate erweitert, die gültige Beziehungen zwischen den Aktivitäten festlegen und so beispielsweise angeben, dass eine bestimmte Aktivität erst ausgeführt werden kann, nachdem eine andere ausgeführt wurde. [LMo7] schlägt eine Syntaxerweiterung für BPEL vor, die es erlaubt einen BPEL-Prozess auf semantische Einschränkungen zu überprüfen und durchzusetzen.

3.4 Änderungsmuster

In [WRMRo7] beschreiben die Autoren eine Reihe von *Änderungsmustern* (*Adaptation patterns*) um die möglichen Änderungen an Geschäftsprozessen darauf zurückführen zu können. Die beschriebenen Änderungsmuster ergänzen eine Reihe von Workflow-Mustern aus [RTHEA05a], [RTHEA05b] und [VDATHKB03] um Muster zur Prozessänderung.

Die beschriebenen Änderungsmuster sind allgemein gehalten und sollten alle möglichen Änderungen an *Workflows* umfassen. Die Muster sind auf ein *Process fragment* (*process fragment*) anwendbar. Ein *Process fragment* kann entweder eine einzelne *Aktivität* sein, oder aus zusammenhängenden Aktivitäten mit jeweils einer einzigen eingehenden und ausgehenden Kante bestehen. In BPEL ist ein *Process fragment* also entweder eine einzelne *Basic activity*, eine *Structured activity* ohne ein- oder ausgehende *Links* oder eine Reihe von *Activities* innerhalb eines `<flow>`-Elements, die so verbunden sind, dass sie über genau einen eingehenden und einen ausgehenden *Link* verfügen. Die Bezeichnung „Aktivität“ bezieht sich im Folgenden in aller Regel sowohl auf eine einzelne Aktivität als auch auf ein *Process fragment*, es sei denn aus dem Kontext wird klar, dass es sich um eine einzelne BPEL *Basic activity* handeln muss.

Die folgende Liste enthält die in [WRMR07] beschriebenen *Adaptation Patterns*. Diese Muster werden in den folgenden Abschnitten ausführlich behandelt.

Hinzufügen und Entfernen

AP1 Einfügen (Insert Process Fragment)

AP2 Entfernen (Delete Process Fragment)

Verschieben und Ersetzen

AP3 Verschieben (Move Process Fragment)

AP4 Ersetzen (Replace Process Fragment)

AP5 Vertauschen (Swap Process Fragments)

AP14 Kopieren (Copy Process Fragment)

Hinzufügen und Entfernen von Sub-Prozessen

AP6 Herauslösen eines Teilprozesses (Extract Sub Process)

AP7 Einbetten eines Teilprozesses (Inline Sub Process)

Kontrollfluss ändern

AP8 Wiederholen (Embed Process Fragment in Loop)

AP9 Parallelisieren (Parallelize Process Fragments)

AP10 Bedingtes Ausführen (Embed Process Fragment in Conditional Branch)

AP₁₁ Übergangsbedingung hinzufügen (Add Control Dependency)

AP₁₂ Übergangsbedingung entfernen (Remove Control Dependency)

AP₁₃ Übergangsbedingung ändern (Update Condition)

Diese Liste enthält sämtliche Änderungsmuster, die auf die Struktur einer Prozessinstanz anwendbar sind [WRRMo8]. Die *Adaptation patterns* AP₃ (Verschieben), AP₄ (Ersetzen), AP₅ (Vertauschen) und AP₁₄ (Kopieren) lassen sich mit Hilfe der Basis-Patterns AP₁ (Einfügen) und AP₂ (Entfernen) realisieren. Die *Adaptation patterns* AP₆ (Herauslösen) und AP₇ (Einbetten) behandeln ebenfalls Änderungen der Prozessstruktur, beziehen sich aber auf Sub-Prozesse. Die übrigen *Adaptation Patterns* behandeln die Modifikation des Kontrollflusses einer Prozessinstanz und werden in dieser Arbeit nicht weiter betrachtet. Im folgenden Abschnitt werden die Änderungsoperationen ausführlicher beschrieben und den relevanten *Adaptation patterns* zugeordnet.

3.5 Operationen im Detail

Dieser Abschnitt enthält die Details zum Ablauf der bei der Instanzmodifikation relevanten Basisoperationen aus der Liste der *Änderungsmuster*. Die Operationen werden speziell bezogen auf *BPEL* betrachtet und zum Teil mit Beispielen in Form von *BPEL*-Code dargestellt. Die Vorgehensweise zur Umsetzung der in diesem Kapitel beschriebenen Operationen wird in Kapitel 5 ausgeführt und die Auswirkung und Überprüfung der Änderungen wird im Abschnitt 3.2 ausführlich besprochen.

3.5.1 Einfügen

Mit der Operation „Einfügen“ (*Adaptation Pattern AP₁*) wird eine einzelne *Aktivität* oder ein *Process fragment* in den Prozess eingefügt. Das *Process fragment* kann auf verschiedene Arten eingefügt werden, die in den folgenden Absätzen behandelt werden.

Serielles Einfügen

Im einfachsten Fall, dem *seriellen Einfügen*, wird ein *Process fragment* zwischen zwei vorhandenen *Activities* eingefügt, die über eine Kante miteinander verbunden sind (Abbildung 3.4). Die beiden vorhandenen Aktivitäten können entweder innerhalb eines `<sequence>`-Elements aufeinander folgend ablaufen, oder innerhalb eines `<flow>`-Elements über einen `<link>` verknüpft sein.

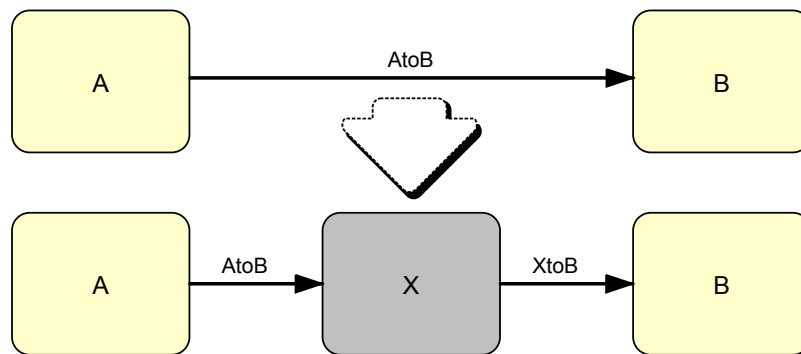


Abbildung 3.4: Serielles Einfügen einer Aktivität

Soll die neue Aktivität in den Verlauf eines `<sequence>`-Elements eingefügt werden, genügt es, die neue Aktivität zwischen den vorhandenen Aktivitäten einzufügen. Eine *Transition condition* oder eine *Join condition* ist innerhalb einer `<sequence>` nicht verfügbar, es sei denn die `<sequence>` ist wiederum selbst Element einer `<flow>`-Aktivität. Für diesen Fall muss die Operation behandelt werden, als wären die vorhandenen Aktivitäten Teil einer `<flow>`-Aktivität.

Soll eine Aktivität zwischen zwei Aktivitäten eingefügt werden, die sich in einem `<flow>` befinden, müssen auch die *Transition conditions* der `<link>`-Elemente sowie die *Join conditions* beachtet werden. Wenn der `<link>` AtoB über eine `<transitionCondition>` verfügt, stellt sich die Frage ob diese Bedingung für die Ausführung der neu eingefügten Aktivität X gültig sein soll, oder ob die neue Aktivität X auch dann ausgeführt werden soll, wenn die Bedingung nicht gültig ist.

Soll die Aktivität X ohne Bedingung unmittelbar nach Aktivität A ausgeführt werden, muss der ursprüngliche Link AtoB mit der zugehörigen `<transitionCondition>` nach dem Einfügen der neuen Aktivität X diese mit Aktivität B verbinden. Zusätzlich muss ein weiterer `<link>` eingefügt werden, der Aktivität A mit Aktivität X verbindet, wie in Codelisting 3.2 dargestellt.

Soll dagegen die Aktivität X unter der ursprünglichen *Transition condition* und *Join condition* ausgeführt werden, wie zuvor die Aktivität B, muss das Ziel des Links AtoB und die zugehörige *Join condition* in die neue Aktivität X verschoben werden. Zusätzlich muss ein weiterer `<link>` eingefügt werden, der Aktivität X mit Aktivität B verbindet. Die *Transition Condition* und die *Join condition* müssen in diesem Fall für Aktivität X gültig sein. Da Aktivität B wiederum ein Nachfolger von Aktivität X ist und damit im Kontrollfluss davon abhängt, sind die Bedingungen auch weiterhin für den Start der Aktivität B gültig. Dieser Fall ist in Codelisting 3.1 dargestellt. Voraussetzung dafür ist allerdings, dass Aktivität X die Bedingung der *Transition condition* nicht wieder ungültig macht.

Der *Link* AtoB kann auch mitsamt *Transition condition* und *Join condition* kopiert und für den neuen *Link* XtoB verwendet werden um sicherzustellen, dass die Bedingungen sowohl für die neu eingefügte Aktivität X, als auch für die vorhandene Aktivität B gültig sind.

Paralleles Einfügen

Die neue Aktivität kann auch parallel zu einer vorhandenen Aktivität eingefügt werden (Abbildung 3.5). In diesem Fall müssen zur eindeutigen Bestimmung der Position der einzufügenden Aktivität entweder sowohl die vorhergehende als auch die nachfolgende Aktivität angegeben werden, oder das *Prozessfragment*, zu dem die Aktivität parallel eingefügt werden soll. Ein parallel auszuführendes Prozessfragment kann in den Prozessverlauf eingefügt werden, indem die vorhandene Aktivität durch ein *<flow>*-Element ersetzt wird, das neben der vorhandenen Aktivität zusätzlich das neue Prozessfragment enthält.

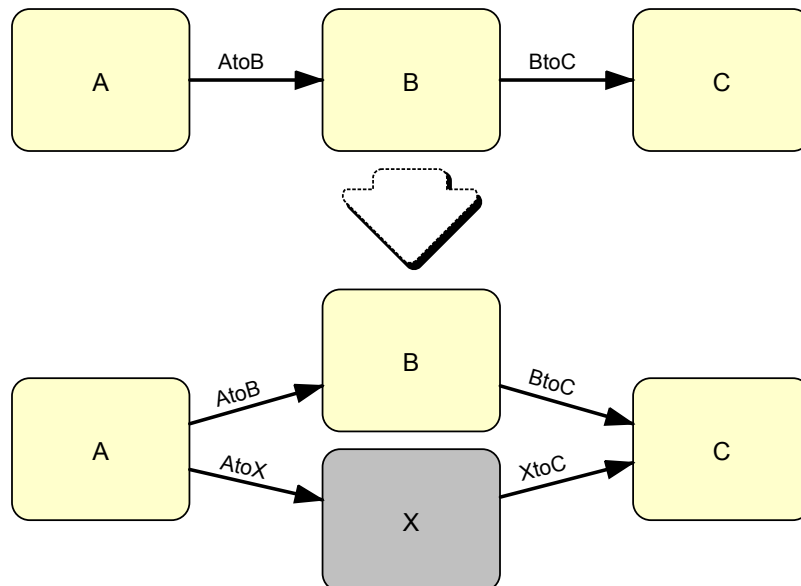


Abbildung 3.5: Paralleles Einfügen einer Aktivität

Einfügen unter Beibehaltung der Join-Condition

Soll ein *Prozessfragment* an einer Position im Prozess eingefügt werden, die Ziel mehrerer *<links>* ist, muss gegebenenfalls auch die *Zusammenführungsbedingung* (*Join condition*) übernommen werden.

Genügt es, wenn die ursprüngliche *Join condition* für das neu eingefügte *Prozessfragment* X gilt, muss das Element *<targets>* mit den enthaltenen Elementen *<target>* und

Codelisting 3.1 Einfügen einer Aktivität in einem <flow> unter Anwendung der ursprünglichen *Transition condition* auf die neu eingefügte Aktivität

```

<flow>
  <links>
    <link name="AtoB"></link> <!-- Urspr. Link A nach B, jetzt A nach X -->
    <link name="XtoB"></link> <!-- Zusätzlich eingefügter Link X nach B -->
  </links>

  <activity name="ActivityA">
    <sources>
      <!-- Unverändert: Ursprung mit Transition condition -->
      <source linkName="AtoB">
        <transitionCondition>$variableA</transitionCondition>
      </source>
    </sources>
  </activity>

  <activity name="ActivityX"> <!-- Neu eingefügte Aktivität -->
    <targets>
      <!-- Die neu eingefügte Aktivität ist Ziel des Links AtoB
            mit der zugehörigen Join cond. -->
      <target linkName="AtoB"></target> ..
      <joinCondition>not($AtoB)</joinCondition>
    </targets>
    <sources>
      <source linkName="XtoB"></source> <!-- Ursprung des neuen Links -->
    </sources>
  </activity>

  <activity name="ActivityB">
    <targets>
      <target linkName="XtoB"></target> <!-- Geändert: XtoB statt AtoB -->
    </targets>
  </activity>
</flow>

```

Codelisting 3.2 Einfügen einer Aktivität in einem <flow> unter Beibehaltung der ursprünglichen *Join condition* für die vorhandene Aktivität B

```

<flow>
  <links>
    <link name="AtoB"></link>  <!-- Urspr. Link A nach B, jetzt X nach B -->
    <link name="AtoX"></link>  <!-- Zusätzlich eingefügter Link A nach X -->
  </links>

  <activity name="ActivityA">
    <sources>
      <source linkName="AtoX"></source>    <!-- Ursprung des neuen Links -->
    </sources>
  </activity>

  <activity name="ActivityX">                <!-- Neu eingefügte Aktivität -->
    <targets>
      <target linkName="AtoX"></target>      <!-- Ziel des neuen Links -->
    </targets>
    <sources>
      <!-- Verschoben: Ursprung mit Transition condition -->
      <source linkName="AtoB">
        <transitionCondition>$variableA</transitionCondition>
      </source>
    </sources>
  </activity>

  <activity name="ActivityB">
    <!-- Aktivität B bleibt unverändert das Ziel des Links AtoB
         mit der zugehörigen Join cond. -->
    <targets>
      <target linkName="AtoB"></target>
      <joinCondition>not($AtoB)</joinCondition>
    </targets>
  </activity>
</flow>

```

`<joinCondition>` aus der Aktivität C in das eingefügte *Prozessfragment* X verschoben werden. Der Übergang von X nach C muss zusätzlich eingefügt werden.

Sollen die Bedingungen der Übergangs- und Zusammenführungsbedingungen weiterhin vor der Ausführung der Aktivität C gültig sein, muss das neu eingefügte Prozessfragment X auf die in Abbildung 3.6 gezeigte Art eingefügt werden. Dabei bleiben die vorhandenen `<link>`-Elemente mit ihren Übergangsbedingungen (`<transitionCondition>`) unverändert. Zusätzlich müssen für die Aktivitäten A und B Übergänge nach X hinzugefügt werden, sowie die `<joinCondition` der Aktivität C um den hinzugefügten Übergang XtoC von Aktivität X nach Aktivität C ergänzt werden. Dabei muss die folgende Bedingung gelten:

$$joinCondition_2 = joinCondition_1 \wedge XtoC$$

wobei $joinCondition_2$ der neuen Zusammenführungsbedingung entspricht und aus der Konjunktion der ursprünglichen Zusammenführungsbedingung $joinCondition_1$ und dem neuen Übergang XtoC gebildet wird. In Codelistung 3.3 wird dieser Vorgang zur Verdeutlichung in BPEL-Code dargestellt.

3.5.2 Entfernen

Mit der Operation „Entfernen“ (*Adaptation Patter AP2*) wird eine *Aktivität* oder ein *Prozessfragment* aus einer Prozessinstanz entfernt. Auch dabei muss zwischen den möglichen Positionen der zu entfernenden Aktivität unterschieden werden. Das Entfernen einer Aktivität kann auf zwei Arten ausgeführt werden: Entweder die Aktivität wird entfernt und damit in der Ausführung übersprungen, oder die Aktivität wird durch eine leere Aktivität ersetzt. Für diese zwei Vorgehensweisen müssen die *Links* und *Conditions* unterschiedlich behandelt werden.

Beim Entfernen einer Aktivität müssen eventuell vorhandene *Datenabhängigkeiten* berücksichtigt werden wie in Kapitel 3.3.3 beschrieben.

Entfernen aus sequentielltem Prozessabschnitt

Im einfachsten Fall soll eine Aktivität aus einer sequentiellen Abfolge von Aktivitäten entfernt werden. Dazu genügt es, statt der zu entfernenden Aktivität die nächste Aktivität auszuführen wie in Abbildung 3.7 dargestellt.

Soll die Aktivität B aus dem Ablauf der Aktivitäten A, B und C entfernt bzw. übersprungen werden, müssen die Aktivitäten A und C mit einem der *Links* AtoB oder BtoC erneut

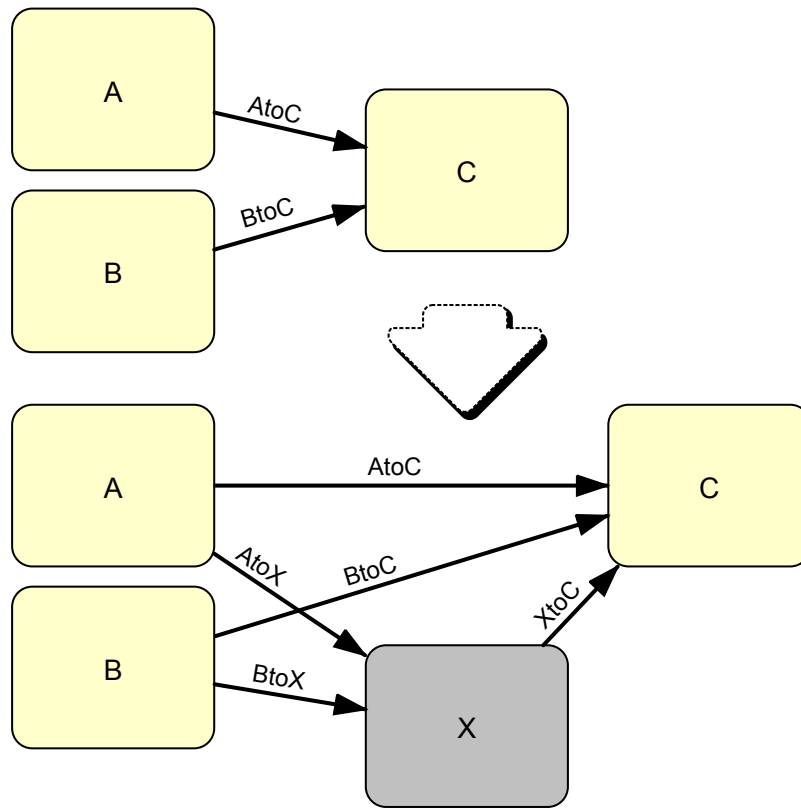


Abbildung 3.6: Einfügen einer Aktivität unter Beibehaltung der joinCondition

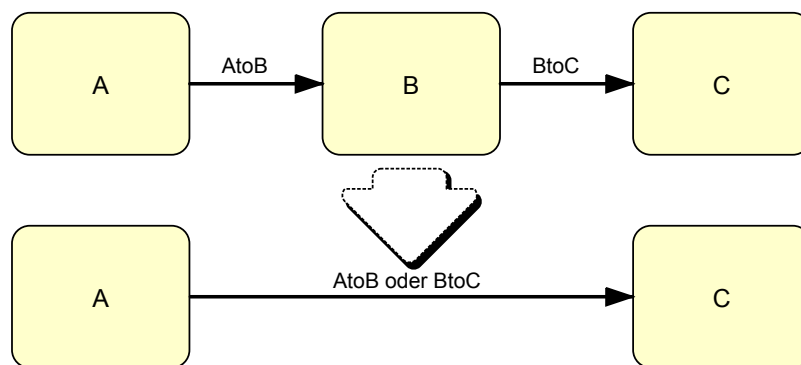


Abbildung 3.7: Entfernen einer Aktivität

Codelisting 3.3 Einfügen einer Aktivität unter Beibehaltung der *join condition*

```

<flow>
  <links>
    <link name="AtoC"></link>
    <link name="BtoC"></link>
    <link name="AtoX"></link>                                <!-- Neu eingefügt -->
    <link name="BtoX"></link>                                <!-- Neu eingefügt -->
    <link name="XtoC"></link>                                <!-- Neu eingefügt -->
  </links>
  <activity name="ActivityA">
    <sources>
      <source linkName="AtoC">
        <transitionCondition>$variableA</transitionCondition>
      </source>
      <source linkName="AtoX"></source>                        <!-- Neu eingefügt -->
    </sources>
  </activity>
  <activity name="ActivityB">
    <sources>
      <source linkName="BtoC">
        <transitionCondition>$varB and $varY</transitionCondition>
      </source>
      <source linkName="BtoX"></source>                        <!-- Neu eingefügt -->
    </sources>
  </activity>
  <activity name="ActivityX">                                <!-- Zusätzlich eingefügte Aktivität -->
    <targets>                                                <!-- auch alle Übergänge sind neu hinzugefügt -->
      <target linkName="AtoX"></target>
      <target linkName="BtoX"></target>
    </targets>
  </activity>
  <activity name="ActivityC">
    <targets>
      <target linkName="AtoC"></target>
      <target linkName="BtoC"></target>
      <target linkName="XtoC"></target>                        <!-- Neu eingefügt -->
      <joinCondition>($AtoC and $BtoC) and $XtoC</joinCondition>
        <!-- Die joinCondition wurde ergänzt um Konjunktion mit XtoC -->
    </targets>
  </activity>
</flow>

```

verknüpft werden. Der zweite *Link* muss entfernt werden. Verfügt der entfernte *Link* über eine *Transition condition* und soll diese erhalten bleiben, muss die Konjunktion aus den *Transition conditions* der beiden *Links* gebildet werden. Wenn $transitionCondition_1$ die *Transition condition* für den Übergang $A \rightarrow B$ ist und $transitionCondition_2$ für den Übergang $B \rightarrow C$, dann gilt für diesen Fall:

$$transitionCondition_1 = transitionCondition_1 \wedge transitionCondition_2$$

Ein Beispiel zum besseren Verständnis dieses Verhaltens ist in Codelisting 3.4 kommentiert. Die Konjunktion der beiden Bedingungen kann allerdings nur angewandt werden, wenn $transitionCondition_2$ nicht von Variablen abhängig ist, die in der entfernten Aktivität geändert wurden. In diesem Fall müsste die *Transition condition* verworfen werden.

Erfolgt das Entfernen der Aktivität durch das Ersetzen mit einer leeren Aktivität, müssen die *Links* der Aktivitäten nicht geändert werden. Auch hierbei gilt jedoch, dass die *Transition condition* der entfernten Aktivität von darin geänderten Variablen abhängig sein kann. Deshalb muss beim Ersetzen einer Aktivität durch eine leere Aktivität auch die *Transition conditions* der ausgehenden *Links* zu ersetzen oder zu entfernen.

Entfernen einer parallel ablaufenden Aktivität

Beim Entfernen einer parallel verlaufenden Aktivität wie in Abbildung 3.8 zu sehen, müssen ebenfalls *Transition conditions* und die *Join conditions* angepasst werden. Wenn im abgebildeten Beispiel die Aktivität D über eine *Join condition* verfügt und die Konjunktion der beiden *Links* $B \rightarrow D$ und $C \rightarrow D$ verlangt, ginge durch ein einfaches Entfernen des *Links* $C \rightarrow D$ möglicherweise eine notwendige Bedingung aus dem entfernten Pfad verloren. Die *Transition conditions* der *Links* $A \rightarrow C$ und $C \rightarrow D$ müssen deshalb gegebenenfalls in die *Transition conditions* des verbleibenden Pfades übernommen werden.

Beim Ersetzen der zu entfernenden Aktivität durch eine leere Aktivität besteht dieses Problem nicht. Dafür muss wie beim Entfernen aus einer *Sequence* ebenfalls darauf geachtet werden, dass die *Transition conditions* der von der entfernten Aktivität ausgehenden *Links* von Variablen abhängen können, die in der entfernten Aktivität C modifiziert wurden. Für diesen Fall muss es möglich sein, die *Transition condition* zu bearbeiten oder zu entfernen.

Codelisting 3.4 Entfernen einer Aktivität aus einem <flow> unter Beibehaltung der ursprünglichen *Transition conditions*

```

<flow>
  <links>
    <!-- Ursprünglicher Link A nach B, jetzt A nach C -->
    <link name="AtoB"></link>
    <!-- <link name="BtoC"></link> Der Link von B nach C wird entfernt -->
  </links>

  <activity name="ActivityA">
    <sources>
      <source linkName="AtoB">
        <!-- Neue Transition condition im Ursprung des Links:
              Konjugation der urspr. Transition conditions -->
        <transitionCondition>$varA and $varB</transitionCondition>
      </source>
    </sources>
  </activity>

  <!-- Aktivität B wurde entfernt. Die ursprüngliche Transition condition $varB
        wurde mit der Transition condition von AtoB konjugiert.
  <activity name="ActivityB">
    <targets>
      <target linkName="AtoB"></target>
    </targets>
    <sources>
      <source linkName="BtoC">
        <transitionCondition>$varB</transitionCondition> // jetzt in AtoB
      </source>
    </sources>
  </activity>
  -->

  <activity name="ActivityC">
    <targets>
      <target linkName="AtoB"></target><!-- war ursprünglich Ziel von BtoC -->
    </targets>
  </activity>
</flow>

```

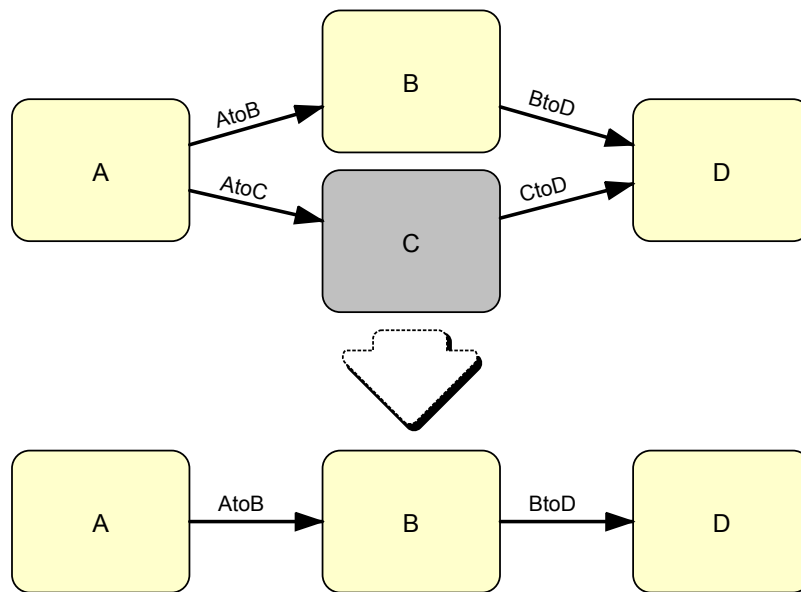


Abbildung 3.8: Entfernen einer parallel verlaufenden Aktivität

Entfernen einer zusammenführenden Aktivität

Wenn eine Aktivität entfernt werden soll, die Ziel mehrerer Übergänge ist, müssen diese Übergänge entsprechend geändert werden. Die auf die entfernte Aktivität folgende Aktivität wird das neue Ziel der Übergänge, wie in 3.9 zu sehen. Die *Join condition* der ursprünglichen Aktivität wird dabei übernommen, da die auf die entfernte Aktivität folgende Aktivität indirekt ebenfalls von der *Join condition* abhängt. Falls der von der gelöschten Aktivität ausgehende *Link* über eine *Transition condition* verfügt, muss diese gegebenenfalls in einen der eingehenden *Links* übernommen werden.

Wird die zu entfernende Aktivität wiederum durch eine leere Aktivität ersetzt, genügt es, wie beim Entfernen einer Aktivität aus einer *Sequence*, darauf zu achten dass die ausgehende *Transition condition* ihre Gültigkeit auch bei einer leeren Aktivität behält.

3.5.3 Verschieben

Das Verschieben (*Adaptation Pattern AP3*) einer Aktivität kann durch die Operationen Entfernen und Einfügen realisiert werden. Die Prozessinstanz muss sich zur erfolgreichen Ausführung dieser Operation in einem Zustand befinden, bei dem keine der beteiligten Aktivitäten bereits gestartet wurde.

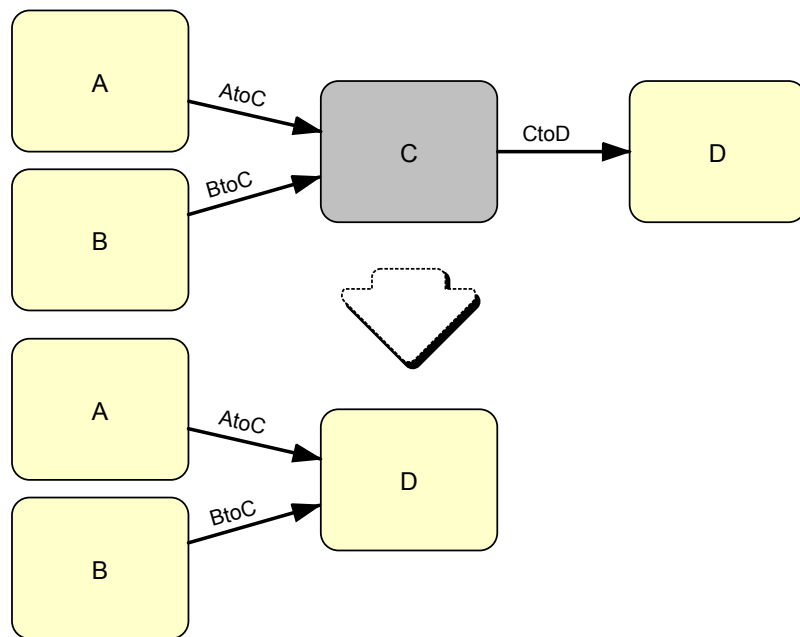


Abbildung 3.9: Entfernen einer zusammenführenden Aktivität

Wie in Abschnitt 3.2.3 dargestellt, darf also weder die ursprüngliche Aktivität, noch eine auf die Zielposition folgende Aktivität bereits aktiv oder beendet sein.

Die Operation *Verschieben*, kann aus den Basis-Operationen *Entfernen* und *Einfügen* zusammengesetzt werden. Dabei gelten die selben Regeln, die für diese Operationen gelten, d. h. die Kontrollabhängigkeiten (*Links*) werden auf die selbe Weise behandelt, wie sie beim Entfernen der Aktivität an einer Position im Prozess behandelt werden und entsprechend auch beim Einfügen an anderer Position.

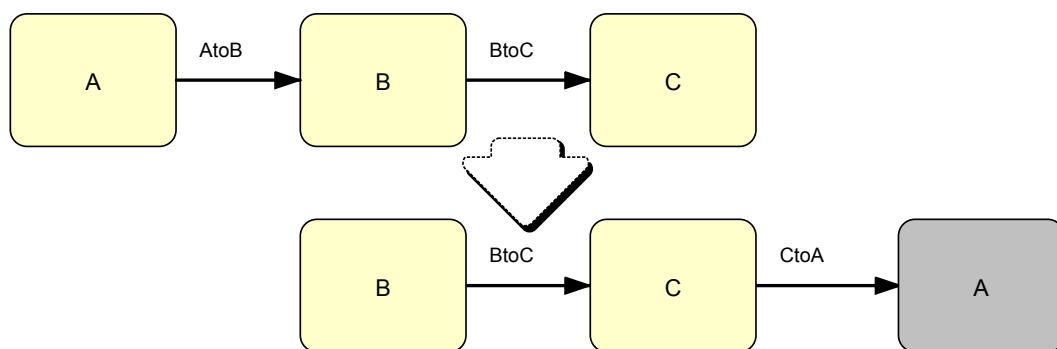


Abbildung 3.10: Verschieben einer Aktivität

Beim Verschieben einer Aktivität müssen vor allem Gültigkeitsbereiche der Variablen, sowie Datenabhängigkeiten der Aktivität beachtet werden.

In dem in Abbildung 3.10 dargestellten Beispiel einer *Sequence* aus den Aktivitäten A, B und C wird die Aktivität A an die Position nach Aktivität verschoben. Damit diese Operation erfolgreich durchgeführt werden kann und nicht zu einem Fehler bei der Ausführung führt, darf Aktivität B nicht von Variablen abhängig sein, die durch Aktivität A initialisiert oder geändert werden.

3.5.4 Ersetzen

Die Operation „Ersetzen“ (*Adaptation Pattern AP4*) einer Aktivität bzw. eines *Process Fragment*, kann durch *Entfernen* des alten und *Einfügen* des neuen Prozessfragmentes erfolgen.

Wie beim *Entfernen* einer Aktivität, müssen dabei die *Datenabhängigkeiten* berücksichtigt werden.

3.5.5 Vertauschen

Beim Vertauschen zweier Prozessfragmente (*Adaptation Pattern AP5*) wie in Abbildung 3.11 gezeigt, müssen, wie beim Verschieben (3.5.3), die Datenabhängigkeiten berücksichtigt werden.

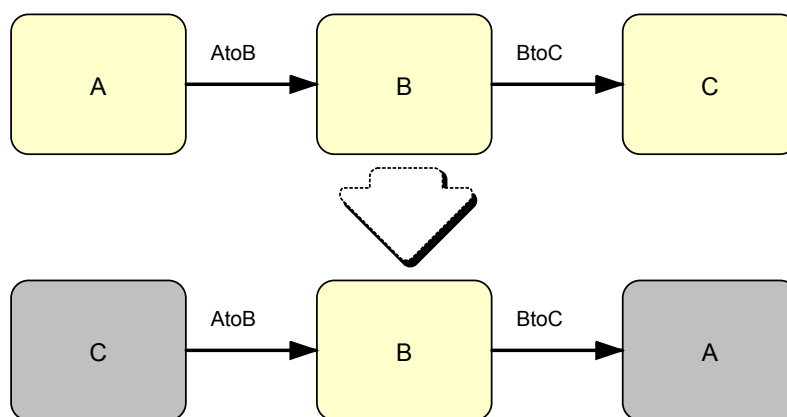


Abbildung 3.11: Vertauschen zweier Aktivitäten

3.5.6 Wiederholen

Soll ein *Process Fragment* mehrfach ausgeführt werden („Wiederholen“ *Adaptation Pattern AP8*), kann es entweder vor der ersten Ausführung in eine entsprechende BPEL-Strukturaktivität wie `<while>`, `<repeatUnit>` oder `<forEach>` eingebunden werden. Befindet sich die Prozessausführung bereits innerhalb des betroffenen *Process Fragment*, muss dieses kopiert und nach der Ausführung der letzten enthaltenen Aktivität eingefügt werden. Wenn es anschließend erneut mehrfach ausgeführt werden soll, kann das *Process Fragment* beim Einfügen in eine Schleife eingebettet werden. Natürlich ist eine entsprechende Abbruchbedingung beim Einfügen einer Schleife notwendig.

Konzepte zur Realisierung

Zur Realisierung der Änderungsoperationen in Apache ODE gibt es grundsätzlich mehrere Möglichkeiten. In diesem Kapitel werden die unterschiedlichen Vorgehensweisen erörtert, beschrieben und anschließend verglichen. Im folgenden Kapitel 5 wird dann die Architektur und Umsetzung eines der Verfahren beschrieben.

4.1 Instanzmodifikation durch Instanzmigration

Ein Verfahren zur Modifikation einer aktiven Prozessinstanz wäre den Status der Prozessinstanz in eine Datei zu exportieren, zu bearbeiten und anschließend als Prozessinstanz eines modifizierten Prozessmodells zu importieren.

Der Begriff „Migration“ ist in Bezug auf Workflows für zwei verschiedene Vorgänge gebräuchlich: Für das Verschieben von Prozessinstanzen einer *Workflow engine* auf zusätzliche Hardware und für das Übernehmen von Workflowinstanzen auf ein neues Prozessmodell. In dieser Arbeit wird der Begriff für das Verschieben einer Instanz auf eine andere *Workflow engine* verwendet. Allerdings wird bei dem beschriebenen Verfahren der *Instanzmodifikation durch Instanzmigration* der Export gleichzeitig auch zur Übernahme der Workflowinstanz auf ein neues Prozessmodell verwendet.

Zur Durchführung dieses Verfahrens ist es notwendig, dass das modifizierte Prozessmodell auf der *Workflow engine* deployed wird. Wenn jedoch nur ein Teil der Prozessinstanzen modifiziert werden soll bedeutet dies, dass die geänderte Version des Prozessmodells unter einem neuen Namen deployed werden muss. Wenn alle Prozessinstanzen geändert werden sollen, kann der neue Prozess unter dem selben Namen deployed werden. Diese Problematik wird in Abschnitt 4.4.2 näher beschrieben.

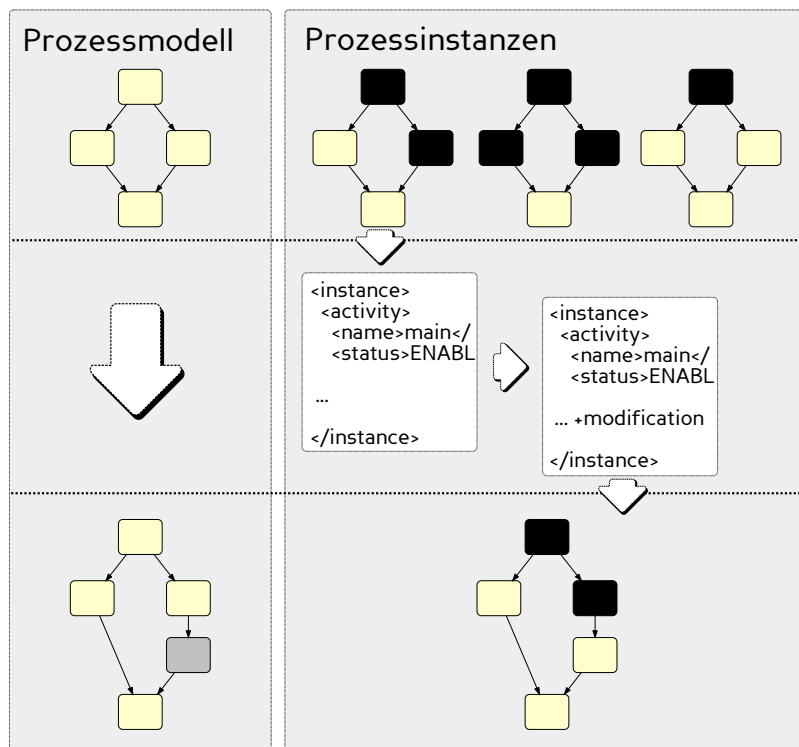


Abbildung 4.1: Darstellung der Instanzmodifikation durch Instanzmigration

Die Vorgehensweise für das Verfahren der *Instanzmodifikation durch Instanzmigration* ist in Abbildung 4.1 zu sehen. Im linken Bereich der Abbildung ist das Prozessmodell dargestellt, das modifiziert werden soll. Dieser Prozess verfügt in der Abbildung über drei Instanzen, die im rechten Bereich abgebildet sind. Die Instanzen befinden sich in unterschiedlichen Zuständen, ausgeführte Aktivitäten sind schwarz dargestellt. Das Vorgehen bei der Instanzmodifikation durch Instanzmigration hat dann den folgenden Ablauf:

1. Zunächst muss definiert werden, welche Änderung am Prozessmodell durchgeführt werden soll und welche Instanzen betroffen sind. Mit den Änderungen muss dann ein neues Prozessmodell erstellt werden.
2. Die Prozessinstanz, die geändert werden soll, muss angehalten werden um den Export zu ermöglichen.
3. Die Prozessinstanz wird in ein XML-Format exportiert. Die exportierten Daten enthalten den vollständigen Zustand der Prozessinstanz inklusive des Zustandes der Aktivitäten und der Inhalte der Variablen.
4. Die exportierten Daten werden so bearbeitet, dass die exportierte Prozessinstanz kompatibel zum neuen Prozessmodell ist. Gegebenenfalls müssen dafür Aktivitäten

eingefügt oder entfernt werden, oder Werte von Variablen müssen geändert werden. Wenn das neue Prozessmodell unter einem neuen Namen auf der *Workflow engine* deployed wurde, muss in den exportierten Daten zudem der Name des Prozesses angepasst werden.

5. Die geänderten Daten werden als Instanz des neuen Prozessmodells wieder eingespielt. Das neue Prozessmodell muss dementsprechend vor diesem Schritt mit einem neuen Namen auf der *Workflow engine* deployed worden sein.
6. Fortsetzen der neuen Instanz auf dem neuen Prozessmodell.

Für das XML-Format wäre auch denkbar, dass damit mehrere Instanzen in einer Exportdatei gespeichert werden. Ein solches Exportformat könnte auch dazu genutzt werden, laufende Prozessinstanzen bei Bedarf auf einen weiteren *Apache ODE*-Server zu verschieben und dort weiter durchzuführen. Zu diesem Zweck ist ein Export-Format für Prozessinstanzen im *Apache ODE*-Projekt in Planung, es gibt zum jetzigen Zeitpunkt jedoch keine Umsetzung [ELU].

Um das XML-Format auch zum Export der Prozessinstanzen auf einen weiteren Workflow-Server zu nutzen, müsste zusätzlich zu den Instanzinformationen auch das Prozessmodell selbst enthalten sein.

Werden beim Exportieren der Prozessinstanz nur die Zustände der Aktivitäten und die Inhalte der Variablen gespeichert, fehlt die Kommunikation mit den aufrufenden und aufgerufenen Partnern, sowie die gespeicherte Eventhistorie (*Audit trail*) des Prozessverlaufs. Die Prozessinstanz ist in *Apache ODE* auch ohne die Kommunikationshistorie ausführbar, der Prozessverlauf kann dann jedoch nicht exakt nachvollzogen werden. In einigen Branchen in denen z. B. sicherheitskritische Prozesse ablaufen, wie in der Luftfahrtindustrie, ist die Speicherung dieses *Audit trails* für viele Jahre vorgeschrieben [LRoo]. Bei einem Export der Prozessinstanz muss dies also berücksichtigt werden. Im folgenden Abschnitt wird eine Lösung vorgestellt, bei der die Prozesskommunikation erhalten bleibt (nicht der gesamte *Audit trail*).

4.2 Instanzmodifikation durch Replay

Eine weitere Möglichkeit der Umsetzung der Instanzmodifikation wäre es, die Webservice-Kommunikation einer Instanz zu exportieren und in ein geändertes Prozessmodell wieder "einzuspielen". In *Apache ODE* wird die Webservice-Kommunikation jeder Prozessinstanz gespeichert und kann somit exportiert werden. Das Vorgehen dabei ist dem Vorgehen bei der *Instanzmodifikation durch Instanzmigration* sehr ähnlich. Statt des Exports des Zustandes der Prozessinstanz wird jedoch die gesamte Webservice-Kommunikation in ein XML-Format

exportiert. Die so exportierten Daten müssen allerdings im Unterschied zum Vorgehen bei der Instanzmigration nicht bearbeitet werden. Das Prozessmodell wird bearbeitet und die Kommunikationsdaten werden als eine neue Prozessinstanz des neuen Prozessmodells wieder eingespielt. Dies setzt voraus, dass keine der durchgeführten Aktivitäten in den Kommunikationsdaten der Prozessinstanz Teil des *Modification trace* der Änderung ist.

1. Zunächst muss definiert werden, welche Änderung am Prozessmodell durchgeführt werden soll und welche Instanzen betroffen sind. Mit den Änderungen muss dann ein neues Prozessmodell erstellt werden.
2. Die Prozessinstanz, die geändert werden soll, muss angehalten werden um den Export zu ermöglichen.
3. Die Prozessinstanz wird in ein XML-Format exportiert. Diese Export-Daten enthalten den Zeitpunkt der Kommunikation, den Name des Prozesses und der Webservice-Methode und den vollständigen Inhalt der ausgetauschten Daten.
4. Wenn das geänderte Prozessmodell unter einem neuen Namen auf der *Workflow engine* deployed wurde, müssen die Kommunikationsdaten entsprechend angepasst werden. Dieser Schritt kann entfallen, wenn keine Prozessinstanzen mehr auf dem alten Prozessmodell aktiv bleiben müssen und das neue Prozessmodell unter dem selben Namen deployed wurde.
5. Die geänderten Daten werden als Instanz des neuen Prozessmodells wieder eingespielt.
6. Fortsetzen der neuen Instanz auf dem neuen Prozessmodell.

Die Daten der Webservice-Kommunikation müssen dabei die Webservice-Operation, den Inhalt der Parameter und den Zeitpunkt der Operation beinhalten. Beim Import der Kommunikationsdaten wird dann das Verhalten des Prozesses nachgespielt. Dabei wird die Kommunikation nach außen unterdrückt und die Aktivitäten des Prozesses werden unter den Bedingungen durchgeführt, die bei der tatsächlichen Operation gültig waren. Zu diesen Bedingungen zählt insbesondere der Zeitpunkt der Ausführung, der bei Aktivitäten wie `<wait>` eine Rolle spielt.

Auch für dieses Vorgehen besteht das Problem, dass nicht zwei Versionen des Prozessmodells unter dem selben Namen zur Verfügung stehen können. Es müssen also entweder alle Instanzen auf das neue Prozessmodell migriert werden, oder der Nutzer des Prozesses muss für die neuen Instanzen auf den neuen Prozess zugreifen.

4.3 Eventbasierte Instanzmodifikation

Die Idee der *eventbasierten Instanzmodifikation* besteht darin, auf Events der zu ändernden Instanz zu reagieren und die entsprechende Modifikation erst dann durchzuführen, wenn der Prozess unmittelbar vor der Ausführung dieser Aktivitäten steht. Die Instanzmodifikation geschieht dann entweder durch einen Eingriff in die *Execution Queue* der *Workflow engine*, oder, zum Beispiel im Fall einer Einfüge-Operation, durch das Ausführen der einzufügenden Aktivität innerhalb des Eventhandlers.

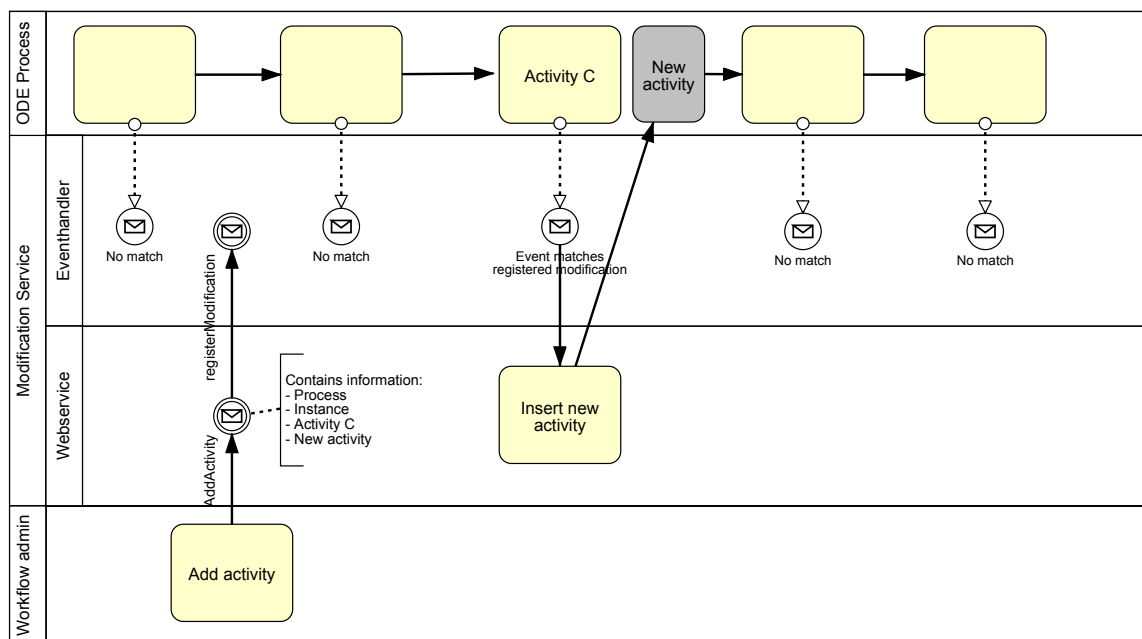


Abbildung 4.2: Ablauf der eventbasierten Modifikation

In Abbildung 4.2 wird der Verlauf einer Modifikation bei der *eventbasierten Instanzmodifikation* dargestellt:

1. Eine Prozessinstanz soll geändert werden. Der Verlauf dieser Instanz ist in der oberen Zeile der Abbildung dargestellt. Jede durchgeführte Aktivität der Instanz löst ein *Event* aus, das vom *Eventhandler* behandelt werden kann. Der *Eventhandler* ist in der zweiten Zeile abgebildet.
2. Vom *Workflow administrator* wird eine zusätzliche Aktivität für die Prozessinstanz registriert. Die Registrierung der neuen Aktivität geschieht über einen *Webservice* und enthält die Daten über die Position der neuen Aktivität im Prozessverlauf, sowie die Definition der Aktivität. Der *Webservice* registriert diese Daten im *Eventhandler*.

3. Sobald ein *Event* auftritt, für das eine Modifikation registriert wurde, wird die Modifikation durchgeführt. In dem abgebildeten Beispiel wäre das Event die erfolgreiche Durchführung der *Activity C*.
4. Die zusätzlich eingefügte Aktivität wird anschließend ausgeführt.
5. Die Prozessinstanz wird weiter ausgeführt. Dabei können erneut *Events* auftreten, auf die vorher eine Modifikation registriert wurde, die dann durchgeführt wird.

Das Entfernen einer Aktivität bei dieser Vorgehensweise ist ebenfalls möglich durch das Ersetzen der Aktivität durch eine leere Aktivität, oder durch das Überspringen der Aktivität. Dabei muss die Modifikation entweder durch ein Event nach der Ausführung der vorangehenden Aktivität ausgelöst werden, oder durch ein Event unmittelbar vor der Ausführung der zu entfernenden Aktivität.

Der Vorteil der *eventbasierten Instanzmodifikation* besteht darin, dass es möglich ist, einen Teil der Instanzen eines Prozessmodelles zu modifizieren, ohne das Prozessmodell ändern zu müssen. Insbesondere ist es dadurch möglich, Instanzen dort „on-the-fly“ zu ändern wo dies nötig ist, während andere Instanzen unverändert weiterlaufen.

Außerdem ist es möglich, eine Änderung auf alle Instanzen eines Prozessmodells anzuwenden. Bereits ausgeführte Aktivitäten der Prozessinstanz lösen keine Events aus. Dadurch bleiben Instanzen unverändert, bei denen die Aktivität, die die Modifikation auslöst, bereits in der Vergangenheit liegt. Dies gilt auch für Schleifen in der Prozessausführung. Eine Schleife kann bereits mehrfach durchlaufen worden sein, bevor die Modifikation in Kraft tritt. Durch das entsprechende Event wird die Modifikation dann beim ersten Schleifendurchlauf danach angewandt.

4.4 Vergleich der Verfahren

Alle drei beschriebenen Verfahren zur Instanzmodifikation können verwendet werden, um die Basisoperationen *Einfügen* und *Entfernen* in einer aktiven Prozessinstanz durchzuführen. Die Verfahren unterscheiden sich jedoch in einigen Details.

4.4.1 Modifikation aller Instanzen (*Migrate*)

Die Modifikation eines Prozessmodells kann im Allgemeinen nicht auf jede aktive Instanz angewandt werden [AJ00], Kapitel 3.2.3. Wenn eine Prozessinstanz so weit ausgeführt wurde, dass der *Modification trace* der Prozessänderung Aktivitäten des *Execution trace*

der Prozessinstanz enthält, kann diese Prozessinstanz nicht auf das neue Prozessmodell übernommen werden.

Wenn alle Prozessinstanzen auf das neue Prozessmodell übernommen werden können, ist es möglich den Prozess unter dem selben Namen weiterzuführen. Bei Anwendung der Verfahren Modifikation durch *Instanzmigration* oder *Replay*, können alle Prozessinstanzen exportiert, und nach dem Aktualisieren des Prozessmodells wieder eingespielt werden. Bei der *eventbasierten Modification* kann die gewünschte Änderung für alle Prozessinstanzen registriert werden. Die Modifikation ist dann also für die drei Verfahren möglich.

Wenn aber nicht alle Prozessinstanzen übernommen werden können sondern einige Prozessinstanzen nach dem alten Prozessmodell weiterbetrieben werden müssen, muss bei den Verfahren der Modifikation durch *Instanzmigration* oder *Replay* das neue Prozessmodell unter einem neuen Namen deployed werden. Das bedeutet, dass sich auch der Webservice-Endpunkt des Prozesses ändert und möglicherweise Clients über diese Änderung informiert werden müssen. Bei einer großen Anzahl von Prozessen ist es nicht praktikabel dies manuell zu tun. Bei Anwendung der *eventbasierten Instanzmodifikation* muss das geänderte Prozessmodell nicht deployed werden, da die Änderungen „on-the-fly“ nur für jene Instanzen vorgenommen werden für die die Änderung noch möglich ist. Allerdings bedeutet dies, dass sich die Durchführung der Modifikation einer Prozessinstanz nur anhand ihres *Audit trails* feststellen lässt.

4.4.2 Modifikation einzelner Instanzen (*Adapt*)

Bei der Modifikation einzelner Instanzen, z. B. zur Behandlung von Sonderfällen tritt grundsätzlich das selbe Problem auf, dass das Prozessmodell unter einem neuen Namen deployed werden muss. Wenn die Zahl der modifizierten Instanzen klein ist, könnte eine manuelle Änderung der Clients jedoch praktikabel sein.

4.4.3 Unterbrechung der Prozessausführung

Die Verfahren *Modifikation durch Instanzmigration* und *Modifikation durch Replay* erfordern eine Unterbrechung der Ausführung der Prozessinstanz durch den Export und erneuten Import. Durch die Registrierung der Modifikationen bei der *eventbasierten Instanzmodifikation* ist dagegen keine Unterbrechung des Prozessverlaufes notwendig. Eine Unterbrechung der Prozessausführung kann trotzdem sinnvoll sein um die weitere Prozessausführung zu verhindern bevor die Modifikation durchgeführt wurde.

4.4.4 Partner links

Für den Fall, dass eine bei der Instanzmodifikation eingefügte Aktivität einen *Partner link* nutzen soll, der in dem ursprünglichen Prozess nicht verfügbar ist, muss dieser Link bei der Änderung mit eingefügt werden. Da bei den Verfahren der *Instanzmigration* und *Replay* neue Prozessmodelle verfügbar sein müssen, können dabei auch neue *Partner links* in das Prozessmodell integriert werden. Bei Anwendung der *eventbasierten Modifikation* wird das Prozessmodell nicht geändert und zur Verwendung neuer *Partner links* müssen diese auf eine andere Art eingefügt werden.

4.4.5 Modifikation bereits durchgeführter Aktivitäten

Bei der Prozessdurchführung eines Workflows können Fälle auftreten, die es notwendig machen, bereits durchgeführte Prozessschritte rückgängig zu machen bzw. erneut durchzuführen. Wenn im Prozess keine *Compensation handlers* vorhanden sind, wäre stattdessen die Behandlung eines solchen Ausnahmefalles mit Hilfe einer speziellen Instanzmodifikation möglich. Dabei muss natürlich die Auswirkung der Änderung auf externe Systeme berücksichtigt werden, wenn Kommunikation stattfand. Bei der *Instanzmodifikation durch Instanzmigration* wäre es möglich, den Status der Aktivitäten, den Inhalt der Variablen und ggf. den *Audit trail* so zu ändern, dass die Durchführung von Aktivitäten beim Import der Daten als nicht getätigt erscheinen. Auch bei der *Instanzmodifikation durch Replay* wäre eine solche Änderung der Instanzdaten möglich, indem die entsprechenden Vorgänge entfernt werden. Bei der *eventbasierten Instanzmodifikation* wäre eine solche Änderung nicht möglich, da dabei nur auf Aktivitäten reagiert werden kann, deren Ausführung in der Zukunft liegt. Die Probleme einer solchen im Prozessmodell nicht vorgesehenen *Compensation* gehen über das Thema dieser Arbeit hinaus. [DDS97] und [KMO98] beschreiben Verfahren zum Umgang mit der *Compensation* bereits ausgeführter Aktivitäten im Ausnahme- oder Fehlerfall.

4.4.6 Zusammenfassung des Vergleiches

In der folgenden Tabelle werden die Vor- und Nachteile der drei beschriebenen Verfahren zur Instanzmodifikation noch einmal zusammengefasst.

	Migration	Replay	Events
Migrate	Ja (Neuer Prozessname wenn nicht alle Instanzen migriert werden können)	Ja (Neuer Prozessname wenn nicht alle Instanzen migriert werden können)	Ja
Adapt	Neuer Prozessname	Neuer Prozessname	Ja
Unterbrechung	Ja	Ja	Nein
Neue <i>Partner links</i>	Ja	Ja	Zus. Aktion
„Compensation“	Theoretisch möglich (hoher Aufwand)	Theoretisch möglich (geringer Aufwand)	Nein

Insbesondere aufgrund der Einschränkung, dass bei den Verfahren *Migration* und *Replay* ein neues Prozessmodell notwendig ist um Änderungen zu übernehmen, wird in dieser Arbeit das Konzept der *eventbasierten Modifikation* als Grundlage der Umsetzung auf *Apache ODE* verwendet.

Entwurf & Implementierung

Dieses Kapitel beschreibt die Umsetzung der *eventbasierten Instanzmodifikation* für die *Workflow engine Apache ODE*. Mit Hilfe der in der Einleitung beschriebenen *Management API* ist es möglich, Informationen über Prozesse und Instanzen abzurufen und Instanzen anzuhalten. Die Modifikation des Prozessverlaufs ist darüber jedoch nicht möglich. Deshalb soll die *Apache ODE* mit der Möglichkeit erweitert werden, Modifikationen an laufenden Prozessinstanzen vorzunehmen. Wie bei der *Management API* soll auch die Verwaltung der Instanzmodifikation über einen *Webservice* möglich sein.

Die grundsätzliche Vorgehensweise bei der *eventbasierten Instanzmodifikation* wurde in Absatz 4.3 vorgestellt. Das genaue Vorgehen bei diesem Verfahren wird in den folgenden Abschnitten ausgearbeitet.

5.1 Architektur

Wie in Abbildung 4.2 angedeutet, wird ein *Modification Service* eingeführt, der über einen *Webservice* gewünschte Änderungen von einem *Webservice-Client* entgegennimmt. Über diesen legt ein *Workflow administrator* die durchzuführenden Änderungen an den Prozessinstanzen fest. Die über den *Webservice* registrierten Änderungen werden in der Schicht des *Event-handlers* gespeichert. Die gespeicherten Änderungen werden dann auf die Prozessinstanz angewandt, wenn das entsprechende *Event* auftritt.

5.1.1 Status einer Prozessinstanz

Die Zustände, die von einer Prozessinstanz durchlaufen werden können, wurden im Abschnitt 3.2.1 besprochen. In *Apache ODE* wurde dieses Modell leicht vereinfacht und umfasst die folgenden Zustände in der Klasse *ProcessState*:

- STATE_NEW - Die Prozessinstanz ist neu und wurde noch nicht gestartet.
- STATE_READY - Die Prozessinstanz wurde gestartet und wartet auf das Aufrufen einer erzeugenden Aktivität.
- STATE_ACTIVE - Die Prozessinstanz wurde mit einer erzeugenden Aktivität aufgerufen und ist aktiv.
- STATE_COMPLETED_OK - Die Prozessinstanz wurde erfolgreich abgearbeitet und ist beendet.
- STATE_COMPLETED_WITH_FAULT - Die Prozessinstanz wurde mit einem unbehandelten Ausnahme beendet.
- STATE_SUSPENDED - Die Prozessinstanz wurde durch einen *Breakpoint* oder durch einen Benutzereingriff angehalten.
- STATE_TERMINATED - Die Prozessinstanz wurde durch eine `<terminate>`-Aktivität oder durch einen Benutzereingriff beendet.

Ein Eingriff in den Prozessverlauf ist nur für Instanzen sinnvoll, die gestartet wurden und noch nicht beendet sind. Das Registrieren von Änderungen am Prozessverlauf ist also für Instanzen mit dem Status STATE_ACTIVE und STATE_SUSPENDED möglich.

5.1.2 Eventmodell & Zustand einer Aktivität

In *Apache ODE* ist ein Event-Modell implementiert, mit dessen Hilfe der Verlauf einer Prozess-Instanz nachverfolgt werden kann. Für diese Arbeit sind die *Events* relevant, die bei der Zustandsänderungen einer Aktivität auftreten.

In *Apache ODE* kann sich eine Aktivität in einem der folgenden Zuständen befinden. Zum Vergleich mit Kapitel 3.2.2 sind die Zustände nach [LRoo] jeweils in Klammer angegeben.

- ENABLED (Executable)
- STARTED (Activated)
- COMPLETED (Completed)
- FAILURE (Terminated)
- DEAD (Dead)

Bei jeder Änderung des Zustandes wird ein *Event* ausgelöst. Dabei kommen die folgenden Events zum Einsatz:

- `ActivityEnabledEvent` – Die Aktivität wurde aktiviert, d.h. die *Join condition* wird ausgewertet.

- `ActivityDisabledEvent` – Die Aktivität wurde Aufgrund einer nicht zutreffenden *Join condition* oder durch eine *Dead path elimination* deaktiviert.
- `ActivityExecStartEvent` – Die Ausführung der Aktivität wird gestartet, d. h. die *Join condition* wurde ausgewertet und ist *true*.
- `ActivityExecEndEvent` – Die Aktivität wurde erfolgreich ausgeführt.
- `ActivityFailureEvent` – Bei der Durchführung der Aktivität trat ein Fehler auf.
- `ActivityRecoveryEvent` – Für die Aktivität wurde über die ManagementAPI eine *Recovery*-Aktion durchgeführt¹.

Apache ODE ermöglicht den Komponenten, einen *EventHandler* auf die auftretenden *Events* zu registrieren. Der *Modification Service* verfügt über einen solchen *EventHandler*. Die entsprechende Modifikation an der Prozessinstanz wird durchgeführt wenn (1.) das entsprechende *Event* (2.) einer bestimmten Aktivität (3.) einer bestimmten Prozessinstanz (4.) eines bestimmten Prozesses auftritt.

Um die Instanzmodifikation auf Basis der *Events* zu ermöglichen, müssen die *Events* in *Apache ODE* leicht angepasst werden. Zu diesem Zweck wurde eine neue abstrakte Klasse `ModifiableEvent` eingeführt, von der alle *Events* abgeleitet werden, mit denen eine Instanzmodifikation möglich sein soll. Die so erweiterten *Events* verfügen über eine zusätzliche Eigenschaft, ein Objekt das eine *Modification* implementiert und das es ermöglicht, eine Rückmeldung vom *EventHandler* zu bekommen. Diese Erweiterung ist nötig, um in der Prozessausführungsschicht der *Workflow engine* eine Rückmeldung vom *Modification Service* erhalten zu können und dort die entsprechende Modifikation vorzunehmen.

Events zum Hinzufügen einer Aktivität

Eine neue Aktivität kann als Reaktion auf jedes *Event* der vorhergehenden Aktivität hinzugefügt werden. Soll im regulären Prozessverlauf eine neue Aktivität hinzugefügt werden, wird diese auf ein `ActivityExecEndEvent` registriert. Um eine zusätzliche Aktivität auszuführen, wenn ein Fehler auftritt, kann diese auf ein `ActivityFailureEvent` registriert werden. Für das `ActivityRecoveryEvent` kann beim Registrieren der zusätzlichen Aktivität angegeben werden, ob die Aktivität bei einer *retry*-, *cancel*- oder *fault*-Aktion oder immer eingefügt werden soll.

¹Über die ManagementAPI kann für eine Aktivität, die sich im Status *Failed* befindet, eine *Recovery*-Aktion durchgeführt werden. Die Activity kann dadurch wiederholt oder abgebrochen werden.

Events zum Entfernen einer Aktivität

Zum Entfernen einer Aktivität, wird im *Modification Service* die zu entfernende Aktivität angegeben und nicht wie beim Hinzufügen die vorhergehende Aktivität. Deshalb können nur die *Events* `ActivityEnabledEvent` oder `ActivityExecStartEvent` verwendet werden. Die Aktivität wird beim Auftreten des entsprechenden *Events* vom *Modification Service* durch eine leere Aktivität ersetzt. Das `ActivityEnabledEvent` wird verwendet, wenn die ursprüngliche *Join condition* nicht ausgewertet werden soll, sondern durch die Standard-*Join condition*, eine einfache Disjunktion der eingehenden *Links*, ersetzt werden soll. Soll die ursprüngliche *Join condition* beibehalten werden, wird das `ActivityExecStartEvent` verwendet um anschließend die leere Aktivität auszuführen. Ob die ursprüngliche *Join condition* ausgewertet werden soll, kann beim Registrieren der zu entfernenden Aktivität im *Modification Service* angegeben werden.

5.1.3 Definition einer neuen Aktivität

Eine neue Aktivität kann im *Modification Service* unter Angabe von BPEL-Code eingefügt werden. Die Operation zum Hinzufügen einer Aktivität verfügt über einen Parameter an den der BPEL-Code der Aktivität als String übergeben werden kann wie in Codelisting 5.1 zu sehen. Die Definition der *Invoke*-Aktivität ist in einen CDATA-Abschnitt eingefasst. Die Notation in der Form `<![CDATA[Inhalt]]>` verhindert, dass der Inhalt ebenfalls als XML interpretiert wird, so dass der Inhaltsteil als String an die *Webservice-Methode* übergeben werden kann.

Codelisting 5.1 Beispielinhalt eines Webservice-Aufrufs zum Hinzufügen einer *Invoke*-Aktivität

```
<InsertNextActivity>
  <ActivityLocation>
    <!-- ActivityLocation: Definition der vorhergehenden Aktivität -->
  </ActivityLocation>
  <NewActivity><![CDATA[<bpel:invoke name="updateSupervision"
    partnerLink="supervisorPartnerLink"
    operation="updateSupervision"
    portType="ns:ModificationClientSoap"
    inputVariable="updateSupervisionRequest"
    outputVariable="updateSupervisionResponse">
    </bpel:invoke>]]></NewActivity>
</InsertNextActivity>
```

Im *Modification Service* wird der so übergebene String mit Hilfe des *BPEL-Compilers* in ein ausführbares Objekt umgewandelt, sobald dieses eingefügt werden soll. Die Übersetzung in ein solches Objekt findet im Kontext der vorhergehenden Aktivität statt. Das heißt bei der Definition der Aktivität stehen die *Variablen*, *Partner links* und *Correlation Sets* zur Verfügung, die in dem *Scope* der vorangehenden Aktivität gültig sind. Im dargestellten Beispiel der eingefügten *Invoke*-Aktivität müsste der *Partner link* `supervisorPartnerLink` mit der entsprechenden Operation und dem *PortType* also im Prozessmodell bereits zur Verfügung stehen. Ebenso müssten die Variablen `updateSupervisionRequest` und `updateSupervisionResponse` im betreffenden *Scope* vorhanden sein. Es ist also über die Schnittstelle zum Hinzufügen einer Aktivität hinaus erforderlich, entsprechende Prozessmodifikationen vornehmen zu können. Dies ist jedoch nicht Teil dieser Arbeit. In dem angegebenen Beispiel wird also vorausgesetzt, dass die verwendeten Elemente in dem entsprechenden Prozess verfügbar sind.

5.1.4 Durchführung der Modifikation

Die tatsächliche Durchführung der Modifikation findet in den Klassen `ACTIVITYGUARD` und `ModificationServiceBpelEventListener` statt. Der *EventHandler* des *Modification Service* greift dabei mit Hilfe der beim Event übergebenen *Modification*-Objekte direkt in die Ausführung einer Aktivität ein.

Hinzufügen einer Aktivität

Die Ausführung von Aktivitäten in der *Apache ODE* geschieht über eine *Execution queue* in der Klasse `JacobVPU`. Um eine neue Aktivität nach einer aktuell ausgeführten Aktivität in den Prozessverlauf einzufügen, wird ein entsprechendes Objekt in diese *Execution queue* eingefügt.

Empfängt der *EventHandler* des *Modification Service* ein *Event* einer Aktivität für die eine entsprechende Modifikation registriert wurde, wird die neue Aktivität mit Hilfe der Methode `JacobVPU.inject()` eingefügt. Nachdem die Aktivität beendet wurde, die das Event ausgelöst hat, wird dann die neu eingefügte Aktivität durchgeführt.

Verfügt die vorangehende Aktivität über ausgehende *Links* werden diese in die neu eingefügte Aktivität übernommen.

Entfernen einer Aktivität

Eine Aktivität wird vom *Modification Service* entfernt, indem sie durch eine leere Aktivität ersetzt wird. Das Vorgehen dabei ist in Abbildung 5.1 dargestellt.

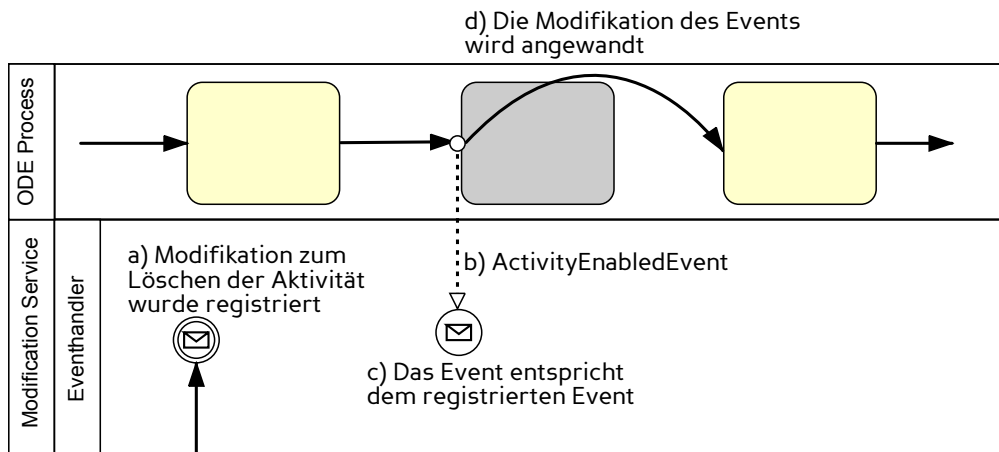


Abbildung 5.1: Vorgehen beim Entfernen einer Aktivität

Zunächst wird (a) die Modifikation zum Entfernen der Aktivität beim *EventHandler* registriert. Beim Auftreten des *ActivityEnabledEvent* der Aktivität die entfernt werden soll (b) reagiert der *EventHandler*. Das mit dem Event übergebenen *Modification*-Objekt wird vom *EventHandler* markiert (c) und enthält daraufhin die Information zum Löschen der Aktivität. Das *Modification*-Objekt des *Events* wird in der *ACTIVITYGUARD*-Klasse vor dem Ausführen überprüft. Enthält es die Information zum Löschen der Aktivität, wird statt der ursprünglichen Aktivität eine leere Aktivität ausgeführt. Die leere Aktivität übernimmt dabei die ausgehenden *Links* der ursprünglichen Aktivität. Beim Registrieren der Modifikation kann angegeben werden, ob die *Transition conditions* der ausgehenden Links ausgewertet werden sollen oder nicht.

Auch die *Join condition* kann durch eine *Standard-Join condition* ersetzt werden, wenn die ursprüngliche *Join condition* für die leere Aktivität nicht gültig sein soll. Wenn die *Join condition* beibehalten werden soll, wird statt des *ActivityEnabledEvent* das *ActivityExecStartEvent* verwendet. Das *Event* *ActivityExecStartEvent* wird erst nach der erfolgreichen Auswertung der *Join condition* ausgelöst.

5.1.5 Fault Handling

Die eingefügten, bzw. die veränderten Aktivitäten oder Fragmente werden im *Scope* ihrer Umgebung ausgeführt und übernehmen dementsprechend das Faulhandling des umgebenden *Scopes*. In einer Änderungsoperation eingefügte Prozessfragmente können *Scopes* und Faulhandler beinhalten. Diese werden auf die selbe Weise behandelt wie im Prozessmodell definierte *Scopes* und Faulhandler.

5.1.6 Persistenz der Änderungen

Um die im *Modification Service* registrierten Änderungen auch nach einem Neustart der *Workflow engine* zu erhalten, müssen die so registrierten Objekte persistent gespeichert werden. Dafür bietet sich die Datenbank an, die auch zur Speicherung der Prozessinstanzen genutzt wird. Beim Starten des *Services* müssen die dort gespeicherten Änderungsobjekte dann wieder in die Liste der beim *Eventhandler* registrierten Modifikationen aufgenommen werden. Die persistente Speicherung der Änderungen ist jedoch nicht Teil der Umsetzung dieser Arbeit.

5.2 Webservice

Der *Webservice* zum Registrieren der gewünschten Änderungen stellt Methoden zum Hinzufügen und Entfernen von Aktivitäten und zur Modifikation von Variablen zur Verfügung. Eine Übersicht über diese Methoden ist in Abbildung 5.2 zu sehen.

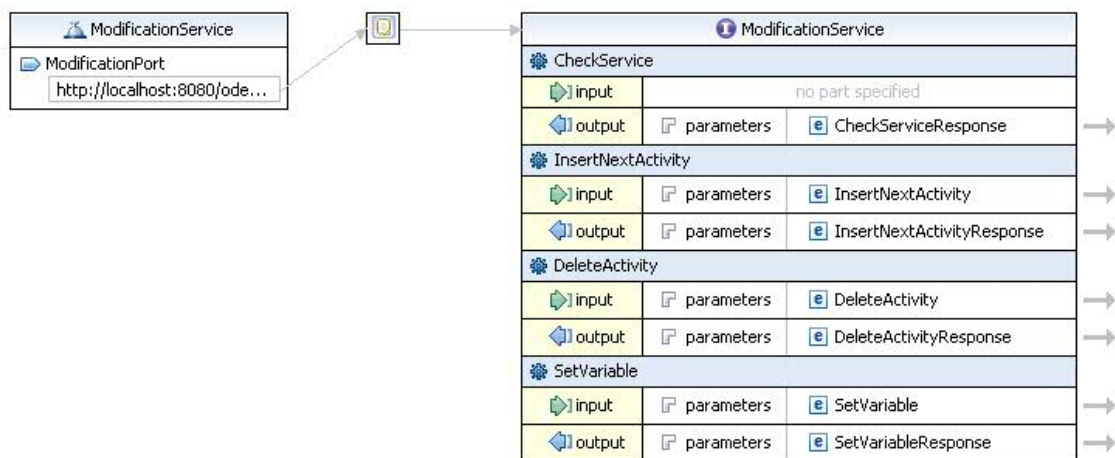


Abbildung 5.2: Übersicht über den Webservice *ModificationService*

5.2.1 Identifikation der Prozessinstanz

Die beim *Modification Service* registrierten Änderungen werden auf bestimmte Prozessinstanzen angewandt. Um die eintreffenden *Events* der gewünschten Instanz zuordnen zu können,

müssen beim Registrieren der Änderung Angaben zur Identifizierung der Prozessinstanz gemacht werden.

BPEL-Prozessinstanzen werden durch den Aufruf einer Aktivität mit der Eigenschaft `createInstance=true` gestartet. Beim Erstellen einer neuen Prozessinstanz, erhält diese Instanz eine eindeutige Nummer innerhalb der *Workflow execution engine*, die *Instance-ID*. Diese ID kann über die *ManagementAPI* abgefragt werden. Beim *Modification Service* kann die *Instanz-ID* zur Bestimmung der zu ändernden Instanzen verwendet werden. Der *Modification Service* ist darauf ausgelegt, eine Änderung auf mehrere Instanzen anwenden zu können. Dazu müssen beim Registrieren der Änderung alle Instanzen angegeben werden, für die die Änderung durchgeführt werden soll. Wird keine Instanz angegeben, wird die Änderung auf alle Instanzen eines Prozesses angewandt.

Ein Prozess verfügt in der *Apache ODE* ebenfalls über eine eindeutige ID. Auch der Name eines Prozesses eines Prozesses ist eindeutig. Wenn keine Einschränkung auf bestimmte Instanzen angegeben wird, muss für eine Änderung die Prozess-ID oder der Prozessname angegeben werden.

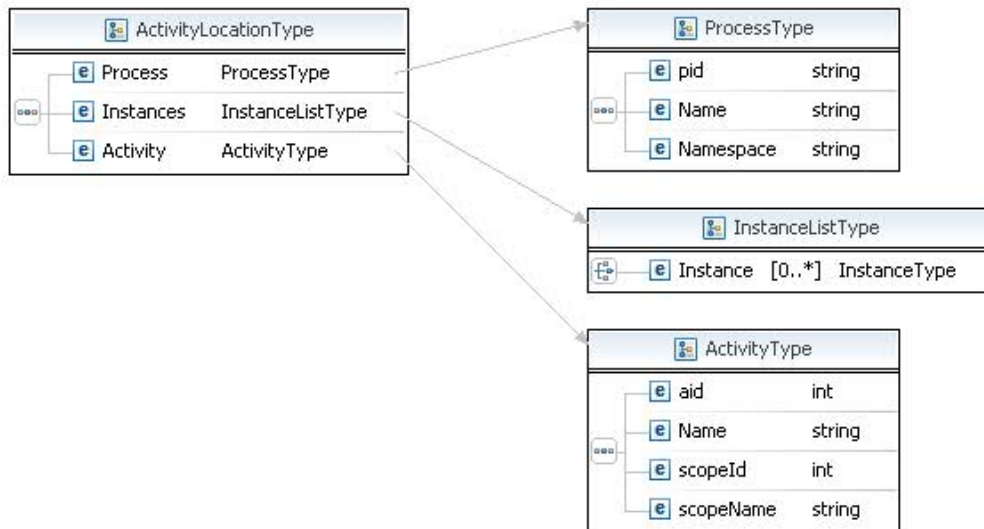
5.2.2 Identifikation einer Aktivität

Zusätzlich zur Bestimmung der betroffenen Prozessinstanzen muss für eine Änderung die Position in der Prozessausführung angegeben werden. Dies geschieht beim *Modification Service* durch die Angabe einer bestimmten Aktivität. Beim Hinzufügen muss die vorhergehende Aktivität angegeben werden, beim Entfernen die zu entfernende Aktivität selbst.

Jede Aktivität in einem Prozess der *Apache ODE* ist Teil eines *Scopes*. Der Prozess ist selbst ein *Scope* und jeder *Scope* erhält eine eindeutige ID. Zudem wird jeder Aktivität eine ID zugewiesen, die über das *Instance management* mit der Methode `getScopeInfoWithActivity()` abgefragt werden kann. Der Name einer Aktivität ist innerhalb eines *Scopes* eindeutig, allerdings muss eine Aktivität nicht zwingend einen Namen haben.

5.2.3 Datentyp zur Lokation der Modifikation

Mit den Informationen zur Identifikation einer Prozessinstanz und zur Identifikation einer Aktivität kann also die Position einer Änderung im Prozessverlauf bestimmt werden. Mit dieser Information kann eine Änderung registriert werden und es ist möglich, die auftretenden *Events* so zu filtern, dass die Änderung an der gewünschten Stelle eingefügt wird. Die Operationen des *Modification Service* zur Modifikation des Prozessverlaufes benötigen diese Angaben. Für den *Webservice* wurde deshalb der Datentyp `ActivityLocationType` erstellt, der diese Daten enthalten kann.

Abbildung 5.3: Darstellung des Datentyps `ActivityLocationType`Abbildung 5.4: Darstellung des Datentyps `InstanceListType`

ActivityLocationType

- Process (ProcessType) – Der Prozess wird über eine dieser Angaben festgelegt. Wenn alle Angaben zum Prozessmodell fehlen, müssen die Instanzen eindeutig festgelegt sein. Wird das Prozessmodell angegeben, müssen keine Instanzen angegeben werden. Die Änderung wird dann auf alle Instanzen des Prozesses angewandt.
 - pid – Die Prozess-ID; fehlt diese Angabe wird stattdessen der Name des Prozesses verwendet.
 - Name – Der Name des Prozesses; fehlt diese Angabe wird der Namespace des Prozesses verwendet.
 - Namespace – Der Namespace des Prozesses; fehlt auch diese Angabe, müssen die Instanzen eindeutig über Instanz-IDs festgelegt sein, sonst wird ein Fehler geworfen.
- Instances (InstanceListType) – Auflistung der Instanzen für die die Änderung durchgeführt werden soll. Es können beliebig viele Instanzen angegeben werden. Wenn keine Instanz angegeben wird, muss ein Prozessmodell angegeben werden und die Änderung wird auf alle Instanzen des angegebenen Prozesses angewandt.
 - Instance (Instance)
 - * iid – Die Instanz-ID einer Instanz; fehlt diese Angabe, wird die *Correlation property* verwendet.
 - * CorrelationProperty – Alternativ zur Instanz-ID kann auch eine *Correlation property* angegeben werden.
- Activity (ActivityType)
 - aid – Die ID der Aktivität².
 - Name – Alternativ zur Activity-ID kann der Name der Aktivität angegeben werden. Dieser ist innerhalb eines Scopes eindeutig.
 - scopeId – Die Scope-ID kann nur angegeben werden, wenn eine einzelne Instanz modifiziert werden soll, da jeder Scope in jeder Instanz eine eigene ID hat.
 - scopeName – Der Name des Scopes kann zur Identifikation des Scopes verwendet werden, wenn mehrere Instanzen geändert werden sollen. Wenn eine Activity-ID angegeben ist, ist die Angabe eines Scopes nicht nötig.

²Die ID der Aktivitäten kann über die Methode `getScopeInfoWithActivity()` im *Instance management* der *Management API* abgerufen werden.

5.2.4 Operation zum Einfügen einer Aktivität

Die Operation zum Einfügen einer Aktivität im *Modification Service* heißt `InsertNextActivity`. Die Parameter werden in der folgenden Liste definiert:

- `ActivityLocation` – Die Bestimmung der Position der Modifikation mit Hilfe des `ActivityLocationType`. Dabei muss die Aktivität angegeben werden, nach der die neue Aktivität eingefügt werden soll.
- `NewActivity` – Ein String mit dem BPEL-Code der neu einzufügenden Aktivität wie im Codelistung 2.1 in Abschnitt 5.1.3 dargestellt.
- `InsertOptions` – Optionen zur Durchführung der Operation
 - `EventType` – Der Typ des Events das zum Einfügen der Aktivität ausgewertet werden soll als String, wie in 5.1.2. Wenn die Angabe fehlt wird als Standard „`ActivityExecEndEvent`“ angenommen.
 - `RecoveryAction` – Für den Eventtyp „`ActivityRecoveryEvent`“ kann zusätzlich die Aktion der *Recovery* angegeben werden – *retry*, *cancel* oder *fault* sind die möglichen Angaben.

Mit dem Aufruf dieser Operation wird die angegebene Modifikation im *Modification Service* registriert. Durchgeführt wird die Änderung erst, wenn eine der im Parameter `ActivityLocation` angegebenen Prozessinstanzen die entsprechende Aktivität ausführt.

5.2.5 Operation zum Entfernen einer Aktivität

Die Operation zum Entfernen einer Aktivität im *Modification Service* heißt `DeleteActivity`. Die Parameter werden in der folgenden Liste definiert:

- `ActivityLocation` – Die Bestimmung der Position der Modifikation mit Hilfe des `ActivityLocationType`. Dabei muss die Aktivität angegeben werden, nach der die neue Aktivität eingefügt werden soll.
- `NewActivity` – Ein String mit dem BPEL-Code der neu einzufügenden Aktivität wie im Codelistung 2.1 in Abschnitt 5.1.3 dargestellt.
- `DeleteOptions` – Optionen zur Durchführung der Operation
 - `EvaluateJoinCondition` – Boolescher Wert zur Angabe darüber, ob die vorhandene *Join condition* der Aktivität ausgewertet werden soll (*true*) oder durch das Standard-Verhalten der *Join condition* (Disjunktion der eingehenden *Links*) ersetzt werden soll (*false*). Wenn die *Join condition* beibehalten werden soll, wird das Entfernen der Aktivität auf das `ActivityExecStartEvent` registriert. Soll die *Join*

condition ersetzt werden, wird die Modifikation auf das `ActivityEnabledEvent` registriert. Ist die Angabe leer, wird als Standard *true* angenommen, d.h. die *Join condition* wird ausgewertet.

- `EvaluateTransitionConditions` – Boolescher Wert zur Angabe darüber, ob die *Transition conditions* der ausgehenden Links ausgewertet werden sollen *true*. Bei Angabe des Wertes *false* werden die *Transition conditions* der ausgehenden Links entfernt, d.h. nach dem Überspringen der Aktivität werden die ausgehenden Links mit dem Status *true* aktiviert.

Wie beim Hinzufügen wird mit dem Aufruf dieser Operation die angegebene Modifikation im *Modification Service* registriert. Durchgeführt wird die Änderung erst, wenn eine der im Parameter `ActivityLocation` angegebenen Prozessinstanzen die entsprechende Aktivität ausführt.

5.2.6 Operation zum Modifizieren von Variablen

Im Unterschied zu den Operationen zur Modifikation des Prozessverlaufes, wird zur Modifikation von Variablen keine Angabe zur Position der Änderung im Prozessverlauf benötigt. Die angegebene Änderung wird stattdessen sofort angewandt. Die Parameter der Operation werden hier beschrieben:

- `scopeId` – Die ID des *Scopes* der die Variable enthält die geändert werden soll. Zur Modifikation einer globalen Variable kann die ID des „Root-Scopes“ angegeben werden.
- `variableName` – Der Name der Variablen. Innerhalb eines *Scopes* ist dieser Name eindeutig.
- `value` – Der Wert der der Variablen zugewiesen werden soll.

Die Modifikation der Variablen erfolgt nicht über die Ausführungsschicht von *Apache ODE*, sondern durch den direkten Zugriff auf die Datenschicht. Der Wert der Variablen wird dadurch in der Datenbank geändert und bei der nächsten Verwendung durch die Prozessinstanz wird der geänderte Wert ausgelesen.

Anwendungsbeispiel

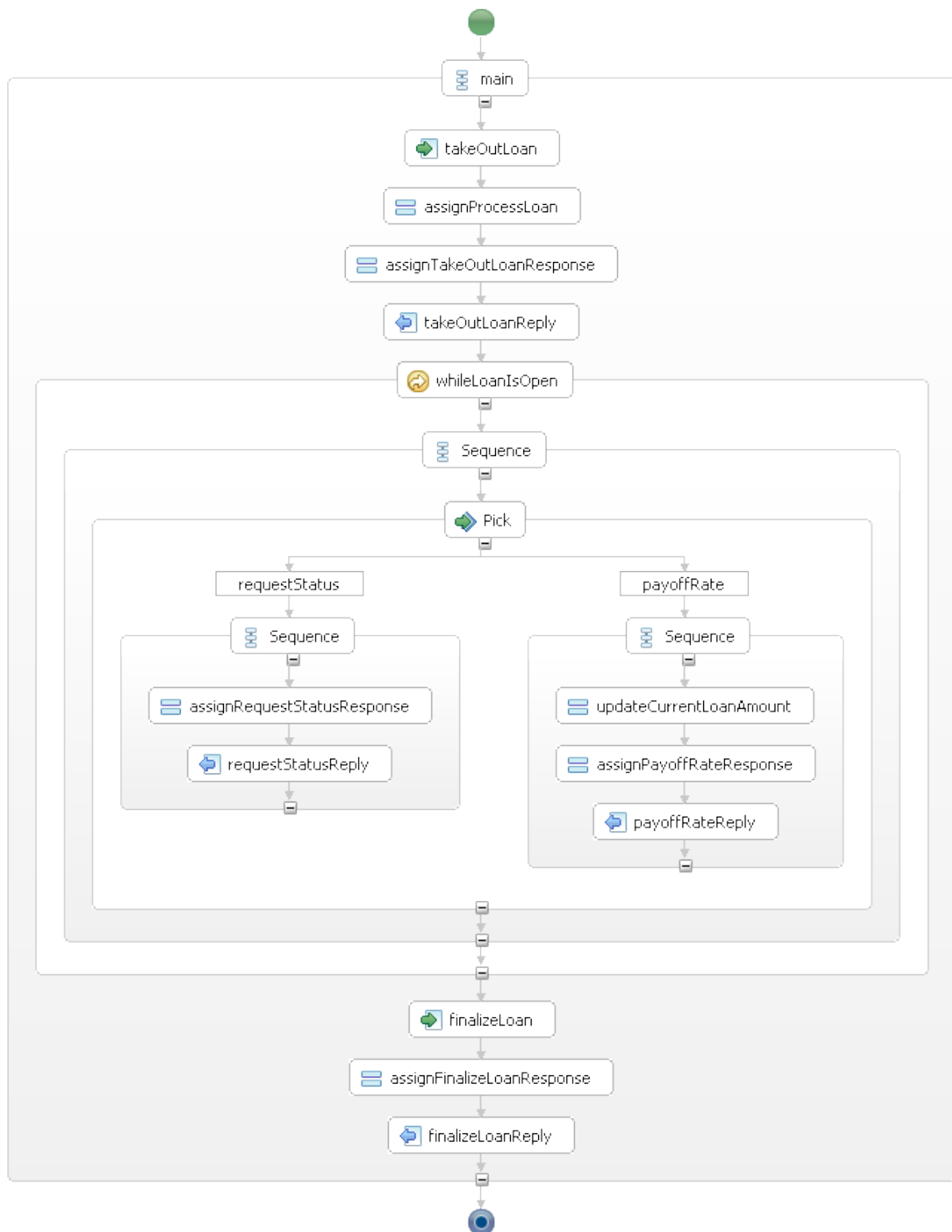
In Kapitel 2.7 wurde ein Beispiel für einen Kreditprozess gegeben, das an dieser Stelle nocheinmal genauer betrachtet werden soll. In der Abbildung 2.2 wurde der Prozess und die Modifikation vereinfacht dargestellt. Abbildung 6.1 zeigt den Prozess wie er in einem BPEL-Editor aussehen könnte. Die zugehörigen Quelltexte der BPEL-Prozessdefinition und der WSDL-Webservice-Schnittstellendefinition sind in Anhang A zu finden. Die vier Prozessschritte des vereinfachten Prozesses sind:

1. Aufnehmen eines Kredites in der Aktivität „Take out loan“
2. Abfrage der aktuellen Kreditdaten mit der Aktivität „Request status“
3. Ratenzahlung eines Kreditbetrages in der Aktivität „Pay installment“ (bzw. „payoffRate“ im BPEL-Beispiel)
4. Abschluss des Kreditvertrages nach der letzten Zahlung in „Finalize loan“

Die entsprechenden Aktivitäten sind auch in dem BPEL-Prozess in Abbildung 6.1 ebenfalls zu finden. Allerdings wurde dieser Prozess durch zusätzliche Aktivitäten zur Zuweisung der Variablen an den Prozess ergänzt damit das Beispiel funktionstüchtig ist und in *Apache ODE* durchgeführt werden kann. Auf *Faulthandler* und *Compensationhandler* wurde in diesem Beispiel verzichtet, um das Beispiel übersichtlich zu halten.

Die Modifikation dieses Workflows besteht nun darin, nach jeder Ratenzahlung eine zusätzliche Aktivität „Update supervision“ durchzuführen. In dieser Aktivität sollen die aktualisierten Kreditdaten an einen Webservice einer zentralen Kreditverwaltung übermittelt werden.

Aus einem BPEL-Prozess kann ein externer Webservice mit Hilfe einer *Invoke*-Aktivität genutzt werden. Diese soll nach dem Ende der Aktivität „Pay installment“ eingefügt werden. Diese Aktivität wird im Beispiel des BPEL-Prozesses mit der <reply>-Aktivität „payoffRateReply“ abgeschlossen. Die Änderung soll auf alle Instanzen des Prozesses *LoanProcess*

Abbildung 6.1: Darstellung des Beispielprozesses *LoanProcess* in einem BPEL-Editor

angewandt werden. Im Aufruf der Webservice-Methode `InsertNextActivity` wird also der Prozess angegeben. Die Angabe der Instanzen bleibt leer. Die Aktivität wird in diesem Beispiel über den Namen der Aktivität und über den Namen des *Scopes* definiert.

Zur Zuweisung der aktuellen Daten an die Variable, die zur Übermittlung der Kreditdaten an den externen Webservice verwendet wird, muss zusätzlich eine `<assign>`-Aktivität eingefügt werden. Damit beide Aktivitäten mit nur einer Operation in den Prozess eingefügt werden können, werden die `<assign>`- und die `<invoke>`-Aktivität in einer `<sequence>`-Aktivität zusammengefasst. Eine grafische Darstellung des Prozessausschnittes der Aktivität „`payoffRateReply`“ mit der anschließend eingefügten `<sequence>`-Aktivität ist in Abbildung 6.2 zu sehen.

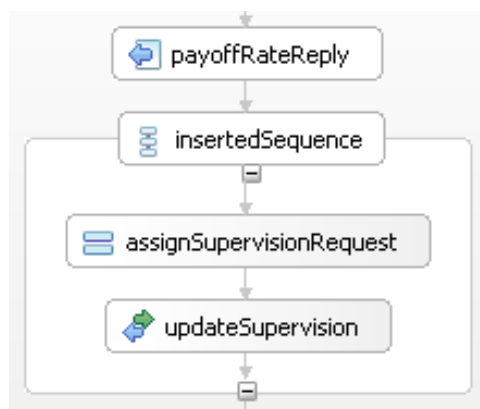


Abbildung 6.2: Darstellung des modifizierten Prozessausschnittes des *LoanProcess*

Ein Beispiel für einen Webservice-Aufruf zum Hinzufügen der beschriebenen `<sequence>` wird im Codelistung 6.1 dargestellt. Als Typ des *Events* wurde hierbei ein `ActivityExecEndEvent` angegeben.

Codelistung 6.1 Beispiel der Webservice-Operation zum Hinzufügen einer *Sequence*

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:mod="http://ode.apache.org/ModificationService/"
    <soapenv:Header/>
    <soapenv:Body>
        <mod:InsertNextActivity>
            <ActivityLocation>
                <Process>
<!-- Zur Angabe des Prozesses wäre einer der Parameter ausreichend -->
                <pid>{http://ode.apache.org/sample/LoanProcess}LoanProcess-1</pid>
                <!-- Fortsetzung auf nächster Seite -->
```

```

        <Name>LoanProcess</Name>
        <Namespace></Namespace>
    </Process>
    <Instances>
<!-- Keine Angabe einer Instanz, Anwendung auf alle Instanzen -->
    </Instances>
    <Activity>
<!-- In diesem Beispiel wird keine Activity-ID verwendet,
    sondern der Name der Aktivität. Alternativ auch die aid angegeben werden,
    die mit Hilfe der Modification API in Erfahrung gebracht werden kann. -->
        <aid></aid>
        <Name>takeOutLoanReply</Name>
        <scopeId></scopeId>
        <scopeName>__PROCESS_SCOPE:LoanProcess</scopeName>
    </Activity>
</ActivityLocation>
<NewActivity><![CDATA[
    <bpel:sequence name="insertedSequence">
        <bpel:assign validate="no" name="assignSupervisionRequest">
            <bpel:copy>
                <bpel:from variable="ProcessLoan"></bpel:from>
                <bpel:to part="parameters" variable="supervisorRequest">
            </bpel:to>
            </bpel:copy>
        </bpel:assign>
        <bpel:invoke
            name="updateSupervision"
            partnerLink="supervisor"
            operation="updateSupervision"
            portType="ns:ModificationClientSoap"
            inputVariable="supervisorRequest"
            outputVariable="supervisorResponse">
        </bpel:invoke>
    </bpel:sequence>]]></NewActivity>
<InsertOptions>
    <EventType>ActivityExecEndEvent</EventType>
    <RecoveryAction></RecoveryAction>
</InsertOptions>
</mod:InsertNextActivity>
</soapenv:Body>
</soapenv:Envelope>

```

Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurden die Möglichkeiten zum *Enforcement* von Änderungen auf aktiven BPEL-Prozessinstanzen untersucht und die Voraussetzungen zur Modifikation einer Instanz wurden betrachtet. Für die möglichen Änderungsoperationen wurden die Schritte erarbeitet, die notwendig sind um die jeweilige Operation erfolgreich durchzuführen. Anschließend wurden Verfahren betrachtet, mit denen die Änderungen auf Prozessinstanzen angewandt werden können. Auf Basis eines dieser Verfahren wurde eine Lösung erstellt, die in der Lage ist, Änderungsoperationen von Prozessinstanzen in der *Workflow engine Apache ODE* durchzuführen.

Apache ODE wurde um einen *Webservice* erweitert, der einen Eingriff in den Prozessverlauf möglich macht. Über diesen *Modification Service* können zusätzliche Aktivitäten in einem Prozess ausgeführt werden. Zudem können Aktivitäten, die nicht ausgeführt werden sollen, übersprungen werden. Die eingefügten Aktivitäten können auch *Structured activities* und damit selbst wiederum Aktivitäten beinhalten. Der Eingriff in die Ausführung der Prozessinstanz erfolgt beim *Modification Service* mit Hilfe von *Events*, so dass der *Modification Service* weitestgehend von der regulären Prozessdurchführung entkoppelt ist.

Schließlich wurde anhand eines verständlichen Beispiels gezeigt, wie der *Modification Service* angewandt werden kann.

7.1 Ausblick

Der im Rahmen dieser Arbeit entwickelte *Modification Service* kann als prototypische Entwicklung zur Grundlage einer praxistauglichen *Workflow engine* mit der Möglichkeit der *Instanzmodifikation* verwendet werden. Da Flexibilität gerade im Bereich von *Workflows* eine wichtige Rolle spielt, wird die Entwicklung in Richtung der Prozess- und Instanzmodifikation weiter gehen. Der vorgestellte *Modification Service* kann zwar den Verlauf eines Prozesses

modifizieren, grundlegende Änderungen am Prozessmodell können damit jedoch nicht durchgeführt werden. Für die praxistaugliche Anwendbarkeit des vorgestellten Ansatzes ist es insbesondere notwendig, eine Möglichkeit zur Einbindung neuer *Partnerlinks* und globaler Variablen in den Prozess zu schaffen.

Zudem sollte es möglich sein, den Prozessverlauf auf eine komfortable Art zu verfolgen. Der *Modification Service* wie er mit dieser Arbeit entwickelt wurde, erlaubt zwar das *Logging* der angewandten Änderungen, aber sowohl für die vom *Modification Service* geänderten Aktivitäten, als auch für den regulären Prozessverlauf wäre eine grafische Darstellung des Prozessverlaufs ein großer Vorteil. Dies würde nicht nur dem *Workflow administrator* die Prüfung der Anwendbarkeit von Änderungen erleichtern, sondern auch Endbenutzern ein besseres Verständnis des Prozessverlaufes ermöglichen.

Um die Anwendung der Prozessmodifikation zu erleichtern, könnte ein BPEL-Editor wie z. B. der bestehende BPEL-Editor für Eclipse¹ so erweitert werden, dass die Unterschiede zwischen zwei Versionen eines Prozessmodells erkannt werden. Über einen erweiterten *Modification Service* könnten diese Unterschiede dann auf Instanzen des älteren Prozessmodells angewandt werden.

¹<http://www.eclipse.org/bpel/>

Anhang A

Codelistings

```
1 <bpel:process name="LoanProcess"
2     targetNamespace="http://ode.apache.org/sample/LoanProcess"
3     suppressJoinFailure="yes"
4     xmlns:tns="http://ode.apache.org/sample/LoanProcess"
5     xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
6     xmlns:ns="http://www.w3.org/2001/XMLSchema">
7
8     <!-- Import the client WSDL -->
9     <bpel:import location="LoanProcessArtifacts.wsdl"
10        namespace="http://ode.apache.org/sample/LoanProcess"
11        importType="http://schemas.xmlsoap.org/wsdl/" />
12
13     <!-- ===== -->
14     <!-- PARTNERLINKS -->
15     <!-- List of services participating in this BPEL process -->
16     <!-- ===== -->
17     <bpel:partnerLinks>
18         <!-- The 'client' role represents the requester of this service. -->
19         <bpel:partnerLink name="client"
20             partnerLinkType="tns:LoanProcess"
21             myRole="LoanProcessProvider"
22             />
23     </bpel:partnerLinks>
24
25     <!-- ===== -->
26     <!-- VARIABLES -->
27     <!-- List of messages and XML documents used within this BPEL process -->
28     <!-- ===== -->
29     <bpel:variables>
30         <bpel:variable name="clientTakeOutLoan"
31             messageType="tns:takeOutLoanRequest"></bpel:variable>
32         <bpel:variable name="clientTakeOutLoanResponse"
33             messageType="tns:takeOutLoanResponse"></bpel:variable>
```

```

31     <bpel:variable name="clientRequestStatus"
           messageType="tns:requestStatusRequest"></bpel:variable>
32     <bpel:variable name="clientRequestStatusResponse"
           messageType="tns:requestStatusResponse"></bpel:variable>
33     <bpel:variable name="clientPayoffRate"
           messageType="tns:payoffRateRequest"></bpel:variable>
34     <bpel:variable name="clientPayoffRateResponse"
           messageType="tns:payoffRateResponse"></bpel:variable>
35     <bpel:variable name="clientFinalizeLoan"
           messageType="tns:finalizeLoanRequest"></bpel:variable>
36     <bpel:variable name="clientFinalizeLoanResponse"
           messageType="tns:finalizeLoanResponse"></bpel:variable>
37     <bpel:variable name="ProcessLoan" type="tns:LoanType"></bpel:variable>
38 </bpel:variables>
39 <bpel:correlationSets>
40     <bpel:correlationSet name="CorrelationSetLoanId"
           properties="tns:ProcessLoanId"></bpel:correlationSet>
41 </bpel:correlationSets>
42
43 <!-- ===== -->
44 <!-- ORCHESTRATION LOGIC -->
45 <!-- Set of activities coordinating the flow of messages across the -->
46 <!-- services integrated within this business process -->
47 <!-- ===== -->
48 <bpel:sequence name="main">
49     <bpel:receive name="takeOutLoan" partnerLink="client"
           createInstance="yes" operation="takeOutLoan" portType="tns:LoanProcess"
           variable="clientTakeOutLoan">
51         <bpel:correlations>
52             <bpel:correlation set="CorrelationSetLoanId" initiate="yes"></bpel:correlation>
53         </bpel:correlations>
54     </bpel:receive>
55
56     <bpel:assign validate="no" name="assignProcessLoan">
57         <bpel:copy>
58             <bpel:from part="parameters" variable="clientTakeOutLoan"></bpel:from>
59             <bpel:to variable="ProcessLoan"></bpel:to>
60         </bpel:copy>
61     </bpel:assign>
62     <bpel:assign validate="no" name="assignTakeOutLoanResponse">
63
64         <bpel:copy>
65             <bpel:from>
66                 <bpel:literal xml:space="preserve"><tns:takeOutLoanResponse
           xmlns:tns="http://ode.apache.org/sample/LoanProcess"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
67 <tns:LoanId></tns:LoanId>
68 <tns:LoanAmount></tns:LoanAmount>

```

```

69 <tns:LoanDate></tns:LoanDate>
70 <tns:LoanCurrentAmount></tns:LoanCurrentAmount>
71 <tns:Person>
72   <tns:Firstname></tns:Firstname>
73   <tns:Lastname></tns:Lastname>
74   <tns:Birthday></tns:Birthday>
75 </tns:Person>
76 </tns:takeOutLoanResponse>
77 </bpel:literal>
78     </bpel:from>
79     <bpel:to variable="clientTakeOutLoanResponse" part="parameters"></bpel:to>
80   </bpel:copy>
81   <bpel:copy>
82     <bpel:from variable="ProcessLoan"></bpel:from>
83     <bpel:to part="parameters" variable="clientTakeOutLoanResponse"></bpel:to>
84   </bpel:copy>
85 </bpel:assign>
86 <bpel:reply name="takeOutLoanReply"
87   partnerLink="client" operation="takeOutLoan" portType="tns:LoanProcess"
88   variable="clientTakeOutLoanResponse"/>
89 <bpel:while name="whileLoanIsOpen">
90   <bpel:condition><![CDATA[(($ProcessLoan/tns:LoanCurrentAmount >
91     0)]]></bpel:condition>
92   <bpel:sequence name="Sequence">
93     <bpel:pick name="Pick"><bpel:onMessage partnerLink="client"
94       operation="requestStatus" portType="tns:LoanProcess"
95       variable="clientRequestStatus">
96       <bpel:sequence>
97         <bpel:assign validate="no" name="assignRequestStatusResponse">
98           <bpel:copy>
99             <bpel:from>
100               <bpel:literal
101                 xml:space="preserve"><tns:requestStatusResponse
102                   xmlns:tns="http://ode.apache.org/sample/LoanProcess"
103                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
104     <tns:LoanId></tns:LoanId>
105     <tns:LoanAmount></tns:LoanAmount>
106     <tns:LoanDate></tns:LoanDate>
107     <tns:LoanCurrentAmount></tns:LoanCurrentAmount>
108     <tns:Person>
109       <tns:Firstname></tns:Firstname>
110       <tns:Lastname></tns:Lastname>
111       <tns:Birthday></tns:Birthday>
112     </tns:Person>
113   </tns:requestStatusResponse>
114   </bpel:literal>
115   </bpel:from>

```

```

109         <bpel:to variable="clientRequestStatusResponse"
110             part="parameters"></bpel:to>
111     </bpel:copy>
112     <bpel:copy>
113         <bpel:from variable="ProcessLoan"></bpel:from>
114         <bpel:to part="parameters"
115             variable="clientRequestStatusResponse"></bpel:to>
116     </bpel:copy>
117 </bpel:assign>
118 <bpel:reply name="requestStatusReply" partnerLink="client"
119     operation="requestStatus" portType="tns:LoanProcess"
120     variable="clientRequestStatusResponse"></bpel:reply>
121 </bpel:sequence>
122 <bpel:correlations>
123     <bpel:correlation set="CorrelationSetLoanId"
124         initiate="no"></bpel:correlation>
125 </bpel:correlations>
126 </bpel:onMessage><bpel:onMessage partnerLink="client"
127     operation="payoffRate" portType="tns:LoanProcess"
128     variable="clientPayoffRate">
129     <bpel:sequence>
130         <bpel:assign validate="no" name="updateCurrentLoanAmount">
131     <bpel:copy>
132         <bpel:from>
133             <![CDATA[$ProcessLoan/tns:LoanCurrentAmount -
134                 $clientPayoffRate.parameters/tns:PayoffAmount]]>
135         </bpel:from>
136         <bpel:to>
137             <![CDATA[$ProcessLoan/tns:LoanCurrentAmount]]>
138         </bpel:to>
139     </bpel:copy>
140     </bpel:assign>
141     <bpel:assign validate="no" name="assignPayoffRateResponse">
142     <bpel:copy>
143         <bpel:from>
144             <bpel:literal
145                 xml:space="preserve"><tns:payoffRateResponse
146                     xmlns:tns="http://ode.apache.org/sample/LoanProcess"
147                     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
148     <tns:LoanId></tns:LoanId>
149     <tns:LoanAmount></tns:LoanAmount>
150     <tns:LoanDate></tns:LoanDate>
151     <tns:LoanCurrentAmount></tns:LoanCurrentAmount>
152     <tns:Person>
153         <tns:Firstname></tns:Firstname>
154         <tns:Lastname></tns:Lastname>
155         <tns:Birthday></tns:Birthday>
156     </tns:Person>

```

```

146 </tns:payoffRateResponse>
147 </bpel:literal>
148         </bpel:from>
149         <bpel:to variable="clientPayoffRateResponse"
150             part="parameters"></bpel:to>
151     </bpel:copy>
152     <bpel:copy>
153         <bpel:from variable="ProcessLoan"></bpel:from>
154         <bpel:to part="parameters"
155             variable="clientPayoffRateResponse"></bpel:to>
156     </bpel:copy>
157 </bpel:assign>
158 <bpel:reply name="payoffRateReply" partnerLink="client"
159     operation="payoffRate" portType="tns:LoanProcess"
160     variable="clientPayoffRateResponse"></bpel:reply>
161 </bpel:sequence>
162 <bpel:correlations>
163     <bpel:correlation set="CorrelationSetLoanId"
164     initiate="no"></bpel:correlation>
165 </bpel:correlations>
166 </bpel:onMessage></bpel:pick>
167 </bpel:sequence>
168 </bpel:while>
169 <bpel:receive name="finalizeLoan" partnerLink="client" operation="finalizeLoan"
170     portType="tns:LoanProcess" variable="clientFinalizeLoan">
171     <bpel:correlations>
172         <bpel:correlation set="CorrelationSetLoanId" initiate="no"></bpel:correlation>
173     </bpel:correlations>
174 </bpel:receive>
175 <bpel:assign validate="no" name="assignFinalizeLoanResponse">
176     <bpel:copy>
177         <bpel:from>
178             <bpel:literal xml:space="preserve"><tns:finalizeLoanResponse
179                 xmlns:tns="http://ode.apache.org/sample/LoanProcess"
180                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
181                 <tns:LoanId></tns:LoanId>
182                 <tns:LoanAmount></tns:LoanAmount>
183                 <tns:LoanDate></tns:LoanDate>
184                 <tns:LoanCurrentAmount></tns:LoanCurrentAmount>
185                 <tns:Person>
186                     <tns:Firstname></tns:Firstname>
187                     <tns:Lastname></tns:Lastname>
188                     <tns:Birthdate></tns:Birthdate>
189                 </tns:Person>
190             </bpel:literal>
191         </bpel:from>
192         <bpel:to variable="clientFinalizeLoanResponse" part="parameters"></bpel:to>

```

```

186         </bpel:copy>
187         <bpel:copy>
188             <bpel:from variable="ProcessLoan"></bpel:from>
189             <bpel:to part="parameters" variable="clientFinalizeLoanResponse"></bpel:to>
190         </bpel:copy>
191     </bpel:assign>
192     <bpel:reply name="finalizeLoanReply" partnerLink="client" operation="finalizeLoan"
193         portType="tns:LoanProcess" variable="clientFinalizeLoanResponse"></bpel:reply>
194 </bpel:sequence>
</bpel:process>

```

Listing A.1: BPEL Prozessdefinition des Beispielprozesses (LoanProcess.bpel)

```

1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3     xmlns:p="http://www.w3.org/2001/XMLSchema"
4     xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
5     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
6     xmlns:tns="http://ode.apache.org/sample/LoanProcess"
7     xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/varprop" name="LoanProcess"
8     targetNamespace="http://ode.apache.org/sample/LoanProcess">
9
10 <!-- ~~~~~
11     TYPE DEFINITION - List of types participating in this BPEL process
12     The BPEL Designer will generate default request and response types
13     but you can define or import any XML Schema type and use them as part
14     of the message types.
15     ~~~~~ -->
16 <property name="ProcessLoanId" type="p:string"
17     xmlns="http://docs.oasis-open.org/wsbpel/2.0/varprop"/>
18 <propertyAlias messageType="tns:finalizeLoanRequest" part="parameters"
19     propertyName="tns:ProcessLoanId"
20     xmlns="http://docs.oasis-open.org/wsbpel/2.0/varprop">
21 <query><![CDATA[tns:LoanId]]></query>
22 </propertyAlias>
23 <propertyAlias messageType="tns:payoffRateRequest" part="parameters"
24     propertyName="tns:ProcessLoanId"
25     xmlns="http://docs.oasis-open.org/wsbpel/2.0/varprop">
26 <query><![CDATA[tns:LoanId]]></query>
27 </propertyAlias>
28 <propertyAlias messageType="tns:takeOutLoanRequest" part="parameters"
29     propertyName="tns:ProcessLoanId"
30     xmlns="http://docs.oasis-open.org/wsbpel/2.0/varprop">
31 <query><![CDATA[tns:LoanId]]></query>
32 </propertyAlias>
33 <propertyAlias messageType="tns:requestStatusRequest" part="parameters"
34     propertyName="tns:ProcessLoanId"
35     xmlns="http://docs.oasis-open.org/wsbpel/2.0/varprop">
36 <query><![CDATA[tns:LoanId]]></query>

```

```

22 </propertyAlias>
23 <types>
24   <schema xmlns="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified"
        elementFormDefault="qualified"
        targetNamespace="http://ode.apache.org/sample/LoanProcess">
25     <element name="takeOutLoan" type="tns:LoanType">
26       </element>
27     <element name="takeOutLoanResponse" type="tns:LoanType">
28       </element>
29     <element name="payoffRate" type="tns:PayoffType">
30       </element>
31     <element name="payoffRateResponse" type="tns:LoanType">
32       </element>
33     <element name="requestStatus" type="tns:LoanType">
34       </element>
35     <element name="requestStatusResponse" type="tns:LoanType">
36       </element>
37     <element name="finalizeLoan" type="tns:LoanType">
38       </element>
39     <element name="finalizeLoanResponse" type="tns:LoanType">
40       </element>
41
42     <complexType name="LoanType">
43       <sequence>
44         <element name="LoanId" type="string"/>
45         <element name="LoanAmount" type="double"/>
46         <element name="LoanDate" type="date"/>
47         <element name="LoanCurrentAmount" type="double"/>
48         <element name="Person" type="tns:PersonType"/>
49       </sequence>
50     </complexType>
51     <complexType name="PayoffType">
52       <sequence>
53         <element name="LoanId" type="string"/>
54         <element name="PayoffAmount" type="double"/>
55       </sequence>
56     </complexType>
57     <complexType name="PersonType">
58       <sequence>
59         <element name="Firstname" type="string"/>
60         <element name="Lastname" type="string"/>
61         <element name="Birthday" type="date"/>
62       </sequence>
63     </complexType>
64   </schema>
65 </types>
66
67 <!-- ~~~~~~

```

```

68     MESSAGE TYPE DEFINITION - Definition of the message types used as
69     part of the port type defintions
70     ~~~~~ -->
71     <message name="LoanProcessRequestMessage">
72         <part element="tns:LoanProcessRequest" name="payload"/>
73     </message>
74     <message name="LoanProcessResponseMessage">
75         <part element="tns:LoanProcessResponse" name="payload"/>
76     </message>
77     <message name="takeOutLoanRequest">
78         <part element="tns:takeOutLoan" name="parameters"/>
79     </message>
80     <message name="takeOutLoanResponse">
81         <part element="tns:takeOutLoanResponse" name="parameters"/>
82     </message>
83     <message name="payoffRateRequest">
84         <part element="tns:payoffRate" name="parameters"/>
85     </message>
86     <message name="payoffRateResponse">
87         <part element="tns:payoffRateResponse" name="parameters"/>
88     </message>
89     <message name="requestStatusRequest">
90         <part element="tns:requestStatus" name="parameters"/>
91     </message>
92     <message name="requestStatusResponse">
93         <part element="tns:requestStatusResponse" name="parameters"/>
94     </message>
95     <message name="finalizeLoanRequest">
96         <part element="tns:finalizeLoan" name="parameters"/>
97     </message>
98     <message name="finalizeLoanResponse">
99         <part element="tns:finalizeLoanResponse" name="parameters"/>
100    </message>
101
102    <!-- ~~~~~
103     PORT TYPE DEFINITION - A port type groups a set of operations into
104     a logical service unit.
105     ~~~~~ -->
106    <portType name="LoanProcess">
107        <operation name="takeOutLoan">
108            <input message="tns:takeOutLoanRequest"/>
109            <output message="tns:takeOutLoanResponse"/>
110        </operation>
111        <operation name="payoffRate">
112            <input message="tns:payoffRateRequest"/>
113            <output message="tns:payoffRateResponse"/>
114        </operation>
115        <operation name="requestStatus">

```

```

116         <input message="tns:requestStatusRequest"/>
117         <output message="tns:requestStatusResponse"/>
118     </operation>
119     <operation name="finalizeLoan">
120         <input message="tns:finalizeLoanRequest"/>
121         <output message="tns:finalizeLoanResponse"/>
122     </operation>
123 </portType>
124
125 <!-- ~~~~~~
126     PARTNER LINK TYPE DEFINITION
127     ~~~~~~ -->
128 <plnk:partnerLinkType name="LoanProcess">
129     <plnk:role name="LoanProcessProvider" portType="tns:LoanProcess"/>
130 </plnk:partnerLinkType>
131
132 <binding name="LoanProcessBinding" type="tns:LoanProcess">
133     <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
134     <operation name="takeOutLoan">
135         <soap:operation
136             soapAction="http://ode.apache.org/sample/LoanProcess/takeOutLoan"/>
137         <input>
138             <soap:body use="literal"/>
139         </input>
140         <output>
141             <soap:body use="literal"/>
142         </output>
143     </operation>
144     <operation name="payoffRate">
145         <soap:operation
146             soapAction="http://ode.apache.org/sample/LoanProcess/payoffRate"/>
147         <input>
148             <soap:body use="literal"/>
149         </input>
150         <output>
151             <soap:body use="literal"/>
152         </output>
153     </operation>
154     <operation name="requestStatus">
155         <soap:operation
156             soapAction="http://ode.apache.org/sample/LoanProcess/requestStatus"/>
157         <input>
158             <soap:body use="literal"/>
159         </input>
160         <output>
161             <soap:body use="literal"/>
162         </output>
163     </operation>

```

```
161     <operation name="finalizeLoan">
162         <soap:operation
163             soapAction="http://ode.apache.org/sample/LoanProcess/finalizeLoan"/>
164         <input>
165             <soap:body use="literal"/>
166         </input>
167         <output>
168             <soap:body use="literal"/>
169         </output>
170     </operation>
171 </binding>
172 <service name="LoanProcessService">
173     <port binding="tns:LoanProcessBinding" name="LoanProcessPort">
174         <soap:address location="http://localhost:8080/ode/processes/LoanProcess"/>
175     </port>
176 </service>
</definitions>
```

Listing A.2: WSDL des Beispielprozesses (LoanProcessArtifacts.wsdl)

Begriffserklärung

Adaptation Pattern *Adaptation patterns* sind Änderungsmuster zur Standardisierung von Änderungsoperationen auf Workflow-Prozessen, siehe Kapitel 3.4

Activity siehe *Aktivität*

Aktivität Eine Aktivität in einem Workflow ist ein Workflow-Schritt. Die Aktivitäten die in einem BPEL-Prozess verfügbar sind, werden in 2.4 beschrieben.

BPEL Web Services Business Process Execution Language, siehe Kapitel 2.4

BPEL-Prozess Ein Workflowmodell, das mit Hilfe der Workflow-Sprache BPEL beschrieben wird und in einer *Workflow engine*, die BPEL unterstützt, ausgeführt werden kann.

Correlation Set Liste von *Properties* zur Identifikation einer Workflow-Instanz bei der Nutzung der Prozess-Operationen

Dead path elimination siehe Kapitel 3.2.2

Deployment Veröffentlichung eines *BPEL-Prozesses* auf einer *Workflow engine*

Event Nachricht, die von einem Objekt eines Programmes gesendet wird.

Eventhandler Ein Objekt in einem Programm, das *Events* eines anderen Objekts empfängt.

Join condition siehe 2.4.7

Link Ein BPEL-Link definiert innerhalb eines *Flows* die Reihenfolge der Ausführung von Aktivitäten, bzw. deren Übergangsbedingungen. Siehe 2.4.5.

Process fragment Ein *Process fragment* ist eine einzelne *Aktivität* oder eine Reihe von *Aktivitäten* mit einer einzigen eingehenden und einer einzigen ausgehenden Kante.

Scope Ein *Scope* in BPEL definiert einen Gültigkeitsbereich von Variablen und kann über *Faulthandler*, *Termination Handler*, *Event Handler* und *Compensation Handler* verfügen.

Transition condition Eine *Transition condition* (*Übergangsbedingung*) legt fest, wann die nachfolgende Aktivität eines *Links* ausgeführt werden soll. Die Bedingung dafür ist im BPEL-Element <source> der ausgehenden Verbindung festgelegt und setzt den Status des Links, siehe 2.4.6

Übergangsbedingung siehe *Transition condition*

Variable siehe 2.4.3

Webservice Ein *Webservice* ist eine Schnittstelle an der ein Software-Dienst zur Verfügung steht.

Workflow administrator Ein *Workflow administrator* in dem in dieser Arbeit verwendeten Sinn hat Zugriff auf die *Workflow engine* und die Berechtigung die darauf laufenden Workflow-Prozesse zu verwalten.

Workflow engine Eine *Workflow engine* ist Software zur Verwaltung und zum betreiben von *Workflows*.

Workflow-Instanz siehe Kapitel 2.3

WSDL (Web Services Description Language) Mit Hilfe der *WSDL* werden *Webservice*-Schnittstellen festgelegt.

Zusammenführungsbedingung siehe *Join condition*

Literaturverzeichnis

- [AJ00] AALST, W.M.P. van d. ; JABLONSKI, S.: Dealing with workflow change: identification of issues and solutions. In: *Computer systems science and engineering* 15 (2000), Nr. 5, S. 267–276 (Zitiert auf den Seiten 21, 22, 23, 26 und 51)
- [Apa] APACHE ODE: *Apache ODE*. <http://ode.apache.org> (Zitiert auf Seite 17)
- [Apa09] APACHE SOFTWARE FOUNDATION: *Apache Axis2*. <http://ws.apache.org/axis2>. Version: 2009 (Zitiert auf Seite 17)
- [DDS97] DU, Weimin ; DAVIS, Jim ; SHAN, Ming-Chien: Flexible specification of workflow compensation scopes. In: *GROUP '97: Proceedings of the international ACM SIGGROUP conference on Supporting group work*. New York, NY, USA : ACM, 1997. – ISBN 0–89791–897–5, S. 309–316 (Zitiert auf Seite 53)
- [ELU] EBERLE, Hanna ; LESSEN, Tammo van ; UNGER, Tobias: *ODE Instance Migration*. Web. <http://www.iaas.uni-stuttgart.de/institut/mitarbeiter/unger/ode/migration.html> (Zitiert auf Seite 48)
- [Hoh05] HOHMANN, Dennis: Anforderungsnahe Realisierung der Laufzeitmodifikationen WS-BPEL basierter Business-Prozesse. (2005) (Zitiert auf Seite 10)
- [KHC⁺05] KARASTOYANOVA, D. ; HOUSPANOSSIAN, A. ; CILIA, M. ; LEYMANN, F. ; BUCHMANN, A.: *Extending BPEL for run time adaptability*. 2005 (Zitiert auf Seite 11)
- [KMO98] KIEPUSZEWSKI, B. ; MUHLBERGER, R. ; ORLOWSKA, M.E.: FlowBack: providing backward recovery for workflow management systems. In: *Proceedings of the 1998 ACM SIGMOD international conference on Management of data* ACM New York, NY, USA, 1998, S. 555–557 (Zitiert auf Seite 53)
- [Ley09] LEYMANN: *Adaptability*. Web. <http://www.iaas.uni-stuttgart.de/lehre/vorlesung/aktuell/vorlesungen/workflow2/materialien/Wfm206Adaptability.pdf>. Version: April 2009. – Vorlesungsunterlagen Workflow Management II (Zitiert auf Seite 23)

- [LMo7] LUCCHI, R. ; MAZZARA, M.: *A pi-calculus based semantics for WS-BPEL*. 2007 (Zitiert auf Seite 30)
- [LRoo] LEYMAN, Frank ; ROLLER, Dieter: *Production workflow: concepts and techniques*. Upper Saddle River, NJ, USA : Prentice Hall PTR, 2000. – ISBN 0-13-021753-0 (Zitiert auf den Seiten 5, 8, 11, 24, 25, 26, 48 und 56)
- [LRDo8] LY, Linh T. ; RINDERLE, Stefanie ; DADAM, Peter: Integration and verification of semantic constraints in adaptive process management systems. In: *Data Knowl. Eng.* 64 (2008), Nr. 1, S. 3-23. <http://dx.doi.org/http://dx.doi.org/10.1016/j.datak.2007.06.007>. – DOI <http://dx.doi.org/10.1016/j.datak.2007.06.007>. – ISSN 0169-023X (Zitiert auf Seite 30)
- [NLKL07] NITZSCHE, J. ; LESSEN, T. van ; KARASTOYANOVA, D. ; LEYMAN, F.: BPEL light. In: *5th International Conference on Business Process Management (BPM)* Springer, 2007 (Zitiert auf Seite 7)
- [OASo7] OASIS WEB SERVICES BUSINESS PROCESS EXECUTION LANGUAGE (WSBPEL) TC: *Web Services Business Process Execution Language Version 2.0*. Web. <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html>. Version: 04 2007 (Zitiert auf den Seiten 5, 9, 11, 12 und 18)
- [RTHEA05a] RUSSELL, N. ; TER HOFSTED, A.H.M. ; EDMOND, D. ; AALST, W.M.P. van d.: Workflow data patterns. In: *Proc. of 24th Int. Conf. on Conceptual Modeling (ER05)*, 2005, S. 353-368 (Zitiert auf Seite 30)
- [RTHEA05b] RUSSELL, N. ; TER HOFSTED, A.H.M. ; EDMOND, D. ; AALST, W.M.P. van d.: Workflow resource patterns, Beta, Research School for Operations Management and Logistics, 2005 (Zitiert auf Seite 30)
- [SO99] SADIQ, Shazia ; ORLOWSKA, Maria: Architectural Considerations in Systems Supporting Dynamic Workflow Modification. In: *In Proceedings of the Workshop on Software Architectures for Business Process Management at the 11th Conf. on Advanced Information Systems Engineering (CaiSE'99)*, 1999 (Zitiert auf den Seiten 22 und 23)
- [VDATHKB03] VAN DER AALST, WMP ; TER HOFSTED, AHM ; KIEPUSZEWSKI, B. ; BARROS, AP: Workflow patterns. In: *Distributed and Parallel Databases* 14 (2003), Nr. 1, S. 5-51 (Zitiert auf Seite 30)
- [W3Co8] W3C: *Extensible Markup Language (XML)*. <http://www.w3.org/XML/>. Version: 2008 (Zitiert auf Seite 12)
- [WRMR07] WEBER, B. ; RINDERLE-MA, S. ; REICHERT, M.: Change Support in Process-Aware Information Systems-A Pattern-Based Analysis. (2007) (Zitiert auf den Seiten 22, 26, 30 und 31)

- [WRRMo8] WEBER, Barbara ; REICHERT, Manfred ; RINDERLE-MA, Stefanie: Change patterns and change support features - Enhancing flexibility in process-aware information systems. In: *Data Knowl. Eng.* 66 (2008), Nr. 3, S. 438–466. <http://dx.doi.org/http://dx.doi.org/10.1016/j.datak.2008.05.001>. – DOI <http://dx.doi.org/10.1016/j.datak.2008.05.001>. – ISSN 0169–023X (Zitiert auf Seite 32)

Alle URLs wurden zuletzt am 25.08.2009 geprüft.

Erklärung

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

(Mattanja Kern)