

Studienarbeit

Grundlagen für die Anpassung der Petri-Netz-Semantik an WS-BPEL 2.0

Peter Laufer

30.05.2006



Zusammenfassung

Die Business Process Execution Language for Web Services (BPEL) ist eine Sprache zur Definition von Geschäftsprozessen als Web Services. In der Industrie nimmt BPEL eine immer stärker werdende Rolle ein, obwohl es bisher keine standardisierte Spezifikation gibt. Für die aktuelle Version BPEL4WS 1.1 [ACD⁺03] liegt eine rein textuelle informelle Spezifikation vor. Um Eigenschaften eines BPEL-Prozesses verifizieren zu können, entwickelte Stahl [Sta05] eine Transformation von BPEL4WS 1.1 in Petrinetze. Als Ergebnis des Standardisierungsprozesses von BPEL wird demnächst die Version WS-BPEL 2.0 verabschiedet werden. Da auch WS-BPEL 2.0 eine textuelle informelle Spezifikation zugrunde liegen wird, wäre eine angepasste Petrinetz-Semantik für Verifikationszwecke weiterhin sehr hilfreich.

Ziel dieser Arbeit ist es deshalb, die Änderungen von WS-BPEL 2.0 im Vergleich zum Vorgänger BPEL4WS 1.1 zu dokumentieren und Vorschläge in Bezug auf die Anpassung der vorhandenen Petrinetz-Semantik zu geben. Die Betrachtungen beziehen sich dabei auf eine Entwurfsfassung der kommenden Spezifikation von WS-BPEL 2.0 vom 16. März 2006 [AAA⁺06].

Inhaltsverzeichnis

1	Einleitung	5
2	BPEL	6
2.1	Was ist BPEL?	6
2.2	Konzepte von BPEL	6
2.3	Probleme von BPEL	8
3	Neuerungen von WS-BPEL 2.0	9
3.1	Ausdrücke und Anfragen	9
3.2	Correlation	10
3.3	Basisaktivitäten	10
3.3.1	Kommunikation	10
3.3.2	Datenmanipulation	12
3.3.3	Sonstige Basisaktivitäten	14
3.4	Strukturierte Aktivitäten	15
3.4.1	Auswahl von Alternativen	15
3.4.2	Schleifen	16
3.5	Link Semantik	19
3.6	Scope	20
3.6.1	Isolierte Scopes	21
3.7	Event Handler	21
3.7.1	Message Events	21
3.7.2	Alarm Events	22
3.8	Fault Handler	22
3.9	Termination Handler	23
3.10	Compensation Handler	23
3.10.1	Standardmäßige Kompensationsreihenfolge	24
3.10.2	Kompensation und isolierte Scopes	25
3.11	Erweiterbarkeit	25
3.12	Abstrakte Prozesse	26
3.12.1	Common Base	27
3.12.2	Profile	28

4	Petrietz-Semantik für WS-BPEL 2.0	29
4.1	Neue Aktivitäten	29
4.2	Correlation	29
4.3	Datenmanipulation	30
4.4	Scope	30
4.5	Event Handler	30
4.6	Fault Handler	30
4.7	Compensation Handler	31
5	Zusammenfassung und Ausblick	32
	Literaturverzeichnis	33

1 Einleitung

Unternehmen, die im Wettbewerb bestehen wollen, müssen stets darauf bedacht sein, ihre Strukturen und Prozesse zu optimieren und wenn möglich effizient zu automatisieren. Der Computer ist dabei in den letzten Jahrzehnten zu einem unverzichtbaren Werkzeug für die Abbildung, Modellierung und Automatisierung von Geschäftsprozessen geworden.

Automatisierbare Geschäftsprozesse werden oft als Web Service implementiert. Unter einem Web Service versteht man dabei eine Prozessbeschreibung (z.B. in BPEL) mit einer Schnittstelle (in WSDL), die unter einem eindeutigen Namen (der Internetadresse des Web Service) erreichbar ist. Oft ist man bestrebt möglichst einfache Web Services zu modellieren, die dann in einem zweiten Schritt zu komplexen Web Services kombiniert werden können. Die Sprache BPEL wird für diese Komposition immer häufiger eingesetzt und nach Abschluss des derzeit laufenden Standardisierungsverfahrens als offizieller Industriestandard gelten.

Die rein informelle, umgangssprachliche, Spezifikation von BPEL macht es jedoch unmöglich, Eigenschaften, wie Korrektheit oder Verklemmungsfreiheit, für diese komponierten Systeme sicherzustellen bzw. nachzuweisen. Ein solcher Nachweis kann nur auf Grundlage einer mathematisch fundierten Semantik erbracht werden. Stahl [Sta05] hat eine solche Semantik für BPEL auf Basis von Petrinetzen entwickelt, mit deren Hilfe sich beliebige in BPEL beschriebene Geschäftsprozesse in Petrinetze überführen und anschließend mit Analysewerkzeugen (wie z.B. Model-Checker) überprüfen lassen.

Das Ziel dieser Arbeit soll es sein, die Veränderungen der neuen BPEL-Version WS-BPEL 2.0 im Vergleich zum Vorgänger BPEL4WS 1.1 zu dokumentieren und Vorschläge in Bezug auf die Anpassung der von Stahl entwickelten Petrinetz-Semantik an die neue BPEL-Version zu geben.

Die Arbeit gliedert sich wie folgt: In Kapitel 2 wird eine kurze Einführung in BPEL gegeben. Der Schwerpunkt der Arbeit liegt in Kapitel 3, wo die Änderungen von WS-BPEL 2.0 ausführlich erläutert werden. Anschließend wird in Kapitel 4 eine Art Arbeitsanleitung zur Anpassung der Petrinetz-Semantik formuliert. Kapitel 5 fasst die Arbeit zusammen und gibt einen Ausblick auf das weitere Vorgehen.

2 BPEL

2.1 Was ist BPEL?

Die *Business Process Execution Language for Web Services* (BPEL) ist eine XML-basierte Sprache zur Definition von Geschäftsprozessen als Web Services.

BPEL ist das Resultat der Vereinigung der beiden Workflow-Sprachen *Web Services Flow Language* (WSFL) [Ley01] und *XLANG* [Tha01]. WSFL wurde von IBM entworfen und basiert auf dem Konzept gerichteter Graphen. XLANG stammt von Microsoft und ist eine block-orientierte Sprache. BPEL verbindet beide Ansätze und bietet daher vielfältige Möglichkeiten zur Beschreibung von Geschäftsprozessen.

Im August 2002 wurde die erste Version von BPEL veröffentlicht. Weitere IT-Firmen (z.B. Oracle) schlossen sich der Initiative an, was im Mai 2003 zu einer überarbeiteten Version 1.1 von BPEL (BPEL4WS 1.1) [ACD⁺03] führte. Um die Akzeptanz von BPEL weiter zu steigern, wurde BPEL im April 2003 an die Organization for the Advancement of Structured Information Standards (OASIS) übergeben. Das von der OASIS gebildete Web Services Business Process Execution Language Technical Committee (WSBPEL-TC) wird demnächst die in weiten Teilen überarbeitete und erweiterte Spezifikation als Standard von BPEL in der Version 2.0 (WS-BPEL 2.0) verabschieden.

2.2 Konzepte von BPEL

Das Kernkonzept von BPEL ist die *Aktivität*. Ein BPEL-Prozess ist genau eine Aktivität, in der weitere Aktivitäten enthalten sind. BPEL unterscheidet dabei zwei Arten von Aktivitäten: Basisaktivitäten und Strukturierte Aktivitäten.

Basisaktivitäten erfüllen die atomaren Aufgaben des Prozesses:

- Kommunikation:
 - Empfang von Nachrichten, mit `<receive>`
 - Beantwortung empfangener Nachrichten, mit `<reply>`
 - Aufruf eines Web Service, mit `<invoke>`
- Datenmanipulation:

- Manipulation von Werten in Variablen, mit `<assign>`
- Sonstige Basisaktivitäten:
 - Nichtstun, mit `<empty>`
 - Warten, mit `<wait>`
 - Signalisieren von Fehlern und Ausnahmen, mit `<throw>`
 - Beenden der Prozessinstanz, mit `<terminate>`
 - Kompensation von abgearbeiteten Aktivitäten, mit `<compensate>`

Basisaktivitäten werden mit Hilfe von Strukturierten Aktivitäten in eine Reihenfolge der Ausführung gebracht:

- Sequentielle Anordnung von Aktivitäten, mit `<sequence>`
- Parallele Anordnung von Aktivitäten, mit `<flow>`
- Auswahl von Alternativen (basierend auf Daten), mit `<switch>`
- Auswahl von Alternativen (basierend auf Nachrichten oder zeitgesteuerten Ereignissen), mit `<pick>`
- Definition von Schleifen, mit `<while>`

BPEL-Prozesse arbeiten nicht in Isolation, sondern interagieren mit anderen Web Services, die man als *Partner* oder *Partner-Prozesse* bezeichnet. Ein Partner ist oftmals sowohl Nutzer eines durch den BPEL-Prozess angebotenen Dienstes als auch selbst Anbieter eines vom BPEL-Prozess benötigten Dienstes. Die Beziehung ist also zumeist bidirektional, insbesondere im Fall von asynchroner Kommunikation. In BPEL werden solche Inter-Prozess-Beziehungen mit Hilfe von so genannten Partner Links beschrieben, in denen z.B. die verwendeten Nachrichtentypen festgelegt werden.

Ein BPEL-Prozess wird in *Instanzen* ausgeführt. Für jeden Partner wird eine neue Instanz erzeugt. Über das Konzept des Correlation Set können Anfragen eines Partners mit seiner entsprechenden BPEL-Prozessinstanz in Verbindung gebracht werden.

Bei der Abarbeitung eines BPEL-Prozesses können Fehler auftreten. Um diese zu behandeln, lassen sich in BPEL Fault Handler definieren. Werden keine Fault Handler spezifiziert, behandelt ein minimaler vordefinierter Fault Handler die auftretenden Fehler.

Oft müssen im Fehlerfall bereits abgeschlossene Aktivitäten in einem Prozess kompensiert werden, wenn sie zum Beispiel in einem transaktionalen Zusammenhang stehen. In BPEL kann dafür ein Compensation Handler definiert werden, mit dessen Hilfe sich diese Kompensationsaktivitäten an geeigneter Stelle spezifizieren lassen.

BPEL-Prozesse lassen sich mit Hilfe von Scopes in Teilprozesse zerlegen. Jeder Scope verfügt dabei über einen eigenen Fault und Compensation Handler. Innerhalb eines Scope können zudem Event Handler definiert werden, über die auf eintreffende Nachrichten oder zeitgesteuerte Ereignisse reagiert werden kann.

2.3 Probleme von BPEL

Die XML-Syntax von BPEL macht es schwer, fehlerfreie Programme zu schreiben, ohne dabei auf die Unterstützung einer Entwicklungsumgebung zurückzugreifen. Eine einheitliche oder gar standardisierte grafische Notation gibt es bisher nicht. Allerdings hat die *Business Process Modelling Notation* (BPMN) [Whi06], die derzeit von der Object Management Group (OMG) standardisiert wird, gute Aussichten sich als Standard im Bereich der grafischen Notation von Geschäftsprozessen (und damit auch BPEL-Prozessen) zu etablieren.

Als Resultat der Vereinigung der beiden grundlegend verschiedenen Konzepte hinter WSFL und XLANG, gibt es mitunter mehrere Repräsentationen für einen Geschäftsprozess in BPEL. Dies zeigt, dass BPEL über redundante Ausdrucksmittel verfügt, was dem Betrachter das intuitive Verständnis eines BPEL-Prozesses erschwert.

Ein weiteres Problem stellt die informelle Spezifikation von BPEL dar. Das heißt, das Dokument beschreibt die Sprache mehr oder weniger umfangreich mit englischem Text und einigen Beispielen. Mehrdeutigkeiten in der Spezifikation führen jedoch zu unterschiedlichem Verhalten von ein und dem selben Prozess in BPEL-Laufzeitumgebungen verschiedener Hersteller. Die mit BPEL beschriebenen Prozessmodelle sind somit bislang weder portabel noch interoperabel. Diesem Umstand soll jedoch mit der neuen erweiterten BPEL-Spezifikation WS-BPEL 2.0 in hohem Maße Rechnung getragen werden, obwohl auch diese rein informell sein wird.

3 Neuerungen von WS-BPEL 2.0

3.1 Ausdrücke und Anfragen

WS-BPEL 2.0 lässt sich in Bezug auf Sprachen zur Definition von Ausdrücken und Anfragen erweitern. Daher besitzt nun jedes WS-BPEL-Sprachelement, in dessen Spezifikation Ausdrücke oder Anfragen definiert werden können, zusätzlich das optionale `expressionLanguage`- bzw. `queryLanguage`-Attribut. In diesem Attribut kann die zur Ausdrucksspezifikation verwendete Sprache angegeben werden.

Im Falle von `expressionLanguage` betrifft dies die folgenden XML-Elemente:

- `<process>`
(generelle Spezifikation der verwendeten Sprache zur Definition von Ausdrücken)
- `<for>`, `<until>`, `<repeatEvery>`
(Definition von Zeitdauer bzw. Zeitpunkten)
- `<condition>`, `<joinCondition>`, `<transitionCondition>`
(Definition von booleschen Bedingungen)
- `<startCounterValue>`. `<finalCounterValue>`, `<branches>`
(Definition von Integer-Ausdrücken)
- `<from>`
(Definition von Ausdrücken zur Belegung von Variablen)

Im Falle von `queryLanguage` betrifft dies die folgenden XML-Elemente:

- `<process>`
(generelle Spezifikation der verwendeten Anfragesprache)
- `<to>`
(Definition des Ziels einer Zuweisung)

Werden diese Attribute nicht spezifiziert, wird in WS-BPEL 2.0 standardmäßig XPath 1.0 [JC99] verwendet, um Ausdrücke und Anfragen auszuwerten.¹

¹Zur Auswertung von XPath 1.0-Ausdrücken siehe WS-BPEL 2.0 Specification, 8.2 Usage of Query and Expression Languages und 8.3 Expressions.

3.2 Correlation

Das `initiate`-Attribut des `<correlation>`-Elements kann zusätzlich zu den Werten "yes" und "no" jetzt auch den Wert "join" annehmen. Die Semantik sieht dabei wie folgt aus: Ist das `initiate`-Attribut auf "join" gesetzt, muss die entsprechende Aktivität versuchen, das Correlation Set zu initialisieren, falls es noch nicht initialisiert worden ist. Ist das Correlation Set initialisiert, aber die Correlation-Konsistenzbedingung verletzt, wird der Fehler `bpws:correlationViolation` geworfen.

Des Weiteren wurde die Semantik für die folgenden Fälle präzisiert:

- Versucht eine Aktivität mit `initiate="yes"` ein Correlation Set zu initialisieren, das bereits initialisiert worden ist, wird ein `bpws:correlationViolation` Fehler geworfen.
- Tritt ein `bpws:correlationViolation` Fehler bei der Abarbeitung einer `<invoke>`-Aktivität auf, so darf dieser erst geworfen werden, wenn die entsprechende Antwort empfangen wurde. In allen anderen Fällen, darf die den `bpws:correlationViolation` Fehler verursachende Nachricht nicht gesendet bzw. empfangen werden.

Die zulässigen Werte für das `pattern`-Attribut des `<correlation>`-Elements wurden von "in" nach "response", von "out" nach "request" und von "in-out" nach "request-response" umbenannt.

3.3 Basisaktivitäten

3.3.1 Kommunikation

Im Gegensatz zu BPEL4WS 1.1 ist das `portType`-Attribut in WS-BPEL 2.0 optional. Der Wert des `portType`-Attributs ergibt sich (wie auch in BPEL4WS 1.1) implizit aus der Kombination des angegebenen `partnerLink` und der durch die Kommunikationsaktivität implizit spezifizierten Rolle.

WS-BPEL 2.0 erleichtert den Umgang mit mehrteiligen WSDL-Nachrichten, durch eine Erweiterung der Kommunikationsaktivitäten um optionale `<toPart>`- und `<fromPart>`-Elemente. Diese dienen dazu, Werte aus BPEL-Variablen als Teile einer Nachricht zu versenden (`<toPart>`-Elemente) bzw. Teile einer empfangenen Nachricht direkt in BPEL-Variablen abzulegen (`<fromPart>`-Elemente). Allerdings gilt folgende Einschränkung: `<receive>`- und `<invoke>`-Aktivitäten, die `<fromPart>`-Elemente enthalten, dürfen kein `variable`- bzw. `inputVariable`-Attribut besitzen, und `<reply>`- und `<invoke>`-Aktivitäten, in denen `<toPart>`-Elemente verwendet werden, dürfen kein `variable`- bzw. `outputVariable`-Attribut besitzen.

Die Syntax der `<toPart>`-Elemente sieht wie folgt aus:

```
<toPart part="ncname" fromVariable="ncname"/>*
```

Mit mehreren `<toPart>`-Elementen lässt sich so eine mehrteilige WSDL-Nachricht mit Daten aus BPEL-Variablen generieren. Dabei belegt jedes `<toPart>`-Element den im `part`-Attribut spezifizierten Teil der WSDL-Nachricht mit dem Inhalt der über das `fromVariable`-Attribut referenzierten BPEL-Variable.

Die Syntax der `<fromPart>`-Elemente ist:

```
<fromPart part="ncname" toVariable="ncname"/>*
```

Die `<fromPart>`-Elemente werden verwendet, um Daten aus eingehenden mehrteiligen WSDL-Nachrichten in individuellen BPEL-Variablen abzulegen. Wird eine WSDL-Nachricht von einer Kommunikationsaktivität empfangen, die `<fromPart>`-Elemente enthält, so werden die einzelnen Teile dieser Nachricht in die entsprechenden BPEL-Variablen kopiert. Dabei belegt jedes `<fromPart>`-Element die im `toVariable`-Attribut spezifizierte BPEL-Variable mit dem Inhalt des über das `part`-Attribut referenzierten Teils der WSDL-Nachricht.

Beispiel: `<invoke>`-Aktivität mit `<toPart>`-Elementen

```
<invoke partnerLink="risikoErmittlung" portType="lms: "
  operation="check" outputVariable="risiko">
  <toPart part="vorname" fromVariable="kundenVorname"/>
  <toPart part="name" fromVariable="kundenName"/>
  <toPart part="betrag" fromVariable="kreditBetrag"/>
</invoke>
```

Die `<invoke>`-Aktivität im obigen Beispiel ist Teil eines Web Service zur Kreditvergabe und ruft einen Partner-Prozess auf, der das Risiko der Kreditvergabe in Bezug auf einen bestimmten Kunden ermittelt. Dieser Partner-Prozess erwartet eine mehrteilige WSDL-Nachricht (bestehend aus "vorname", "name" und "betrag"), die hier mit Hilfe der `<toPart>`-Elemente aus Daten von BPEL-Variablen zusammengestellt wird. Die Antwort auf die Anfrage wird in der Variable "risiko" abgelegt und könnte alternativ zur Angabe des Attributs `outputVariable` auch mittels `<fromPart>`-Elementen einzelnen BPEL-Variablen zugewiesen werden.

Aktivitäten, die Teil eines asynchronen Nachrichtenaustauschs sind, können in WS-BPEL 2.0 per optionalem Attribut `messageExchange` mit einander assoziiert werden². Dazu zählen: `<receive>`- und `<reply>`-Aktivitäten, sowie `<onEvent>`-Zweige eines Event Handlers bzw. `<onMessage>`-Zweige einer `<pick>`-Aktivität. Dies ist insbesondere dann erforderlich, wenn es während einer parallelen Ausführung (z.B. innerhalb einer `<flow>`-Aktivität) zu mehreren Antwortnachrichten für eine Kombination aus `partnerLink` und `operation` kommen kann. Anhand des Tupels `partnerLink`, `operation` und `messageExchange` kann so eine empfangene Anfrage mit einer `<reply>`-Aktivität assoziiert werden. Der Wert des Attributs `messageExchange` muss dabei mit einem `name`-Attribut eines zuvor deklarierten `<messageExchange>`-Elements übereinstimmen.

²Siehe WS-BPEL 2.0 Specification, 10.4 Providing Web Service Operations.

<messageExchange>-Elemente können innerhalb eines Scope bzw. auf Prozessebene deklariert werden und besitzen folgende Syntax:

```
<messageExchanges>?
  <messageExchange name="ncname"/>+
</messageExchanges>
```

Es ist erlaubt, den gleichen Wert für das Attribut `messageExchange` in mehreren zum selben Zeitpunkt unbeantworteten Empfangsaktivitäten zu nutzen, solange sich deren Kombination aus `partnerLink` und `operation` jeweils von einander unterscheidet. Beim Start einer weiteren Kommunikationsaktivität, mit der selben Kombination aus `messageExchange`, `partnerLink` und `operation`, wird der Fehler *bpws:conflictingRequest* geworfen. Kann die, zu einer <reply>-Aktivität assoziierte, unbeantwortete Empfangsaktivität nicht ermittelt werden, wird der Fehler *bpws:missingRequest* geworfen.

3.3.2 Datenmanipulation

Variablen können mit Hilfe der <assign>-Aktivität mit neuen Werten belegt werden. Diese Werte können entweder aus anderen Variablen stammen oder Ergebnisse ausgewerteter Ausdrücke bzw. Literale sein. Abgesehen von der leicht veränderten Syntax der <assign>-Aktivität³ beschreibt die Spezifikation von WS-BPEL 2.0 genaue Anforderungen an die XML-Datenmanipulationen der <copy>-Elemente⁴. Zusätzlich zu den <copy>-Elementen lässt sich die <assign>-Aktivität über <extensibleAssign>-Elemente um weitere Operationen zur Datenmanipulation (aus anderen XML-Namensräumen) erweitern.

Die <assign>-Aktivität ließe bei standardmäßiger Verwendung von XPath 1.0 zur Definition der Ausdrücke keine komplexen XML-Transformationen zu. Die Spezifikation von WS-BPEL 2.0 beschreibt daher eine XPath 1.0 Erweiterungsfunktion *bpws:doXslTransform()*, die diesen Umstand behebt und von standardkonformen Implementationen unterstützt werden muss.⁵

Beispiel: XML-Transformation mit *bpws:doXslTransform()*

```
<sequence>
  <invoke ... outputVariable="A" .../>
  <assign>
    <from>
      <expression>
        bpws:doXslTransform("urn:stylesheets:A2B.xsl", $A)
      </expression>
    </from>
```

³Siehe WS-BPEL 2.0 Specification, 8.4 Assignments.

⁴Näheres siehe WS-BPEL 2.0 Specification, 8.4.1 Selection Results of Copy Operations und 8.4.2 Replacement Logic of Copy Operations.

⁵Für Einzelheiten zu *bpws:doXslTransform()* siehe WS-BPEL 2.0 Specification, 8.4 Assignment.

```

        <to variable="B"/>
    </assign>
    <invoke ... inputVariable="B" .../>
</sequence>

```

Im obigen Beispiel wird zuerst ein Web Service aufgerufen und das Resultat dieses Aufrufs in der Variable "A" gespeichert. Die Daten in der Variable "A" sollen nun als Eingabe für den Aufruf eines zweiten Web Service dienen. Zuvor müssen diese Daten hier jedoch transformiert werden. Dies geschieht über den Aufruf von *bpws:doXsltTransform()*. Der Funktion werden die Transformationsregeln (in Form eines XSLT-Dokuments) und die Daten aus Variable "A" übergeben. Das Ergebnis der Transformation wird der Variable "B" zugewiesen, die dann als Eingabe für den Aufruf des zweiten Web Service verwendet werden kann.

Die neue Spezifikation verlangt, dass jede `<assign>`-Aktivität atomar abgearbeitet wird, d.h. für die Dauer der Ausführung muss eine `<assign>`-Aktivität behandelt werden, als wäre sie die einzige Aktivität im Prozess. Tritt während der Abarbeitung ein Fehler auf, müssen alle Zielvariablen in ihren Zustand vor Beginn der Aktivität zurückgesetzt werden. Dies gilt unabhängig davon, wieviele Zuweisungen die gesamte `<assign>`-Aktivität umfasst.

WS-BPEL 2.0 stellt die neue Basisaktivität `<validate>` zur Verfügung, um explizit testen zu können, ob Variablen bezüglich ihrer XML-Definition mit zulässigen Werten belegt sind.

Die Syntax der `<validate>`-Aktivität sieht wie folgt aus:

```

<validate variables="ncnames-list" standard-attributes>
    standard-elements
</validate>

```

Über das Attribut `variables` kann eine Liste der zu validierenden Variablen angegeben werden. Die einzelnen Bezeichner werden dabei durch Leerzeichen von einander getrennt. Schlägt die Validierung von mindestens einer der zu prüfenden Variablen fehl, wird der Fehler *bpws:invalid-Variables* geworfen.

Die `<assign>`-Aktivität besitzt nun das optionale Attribut `validate`. Es bietet die Möglichkeit, die von der Aktivität durchgeführten Manipulationen an den betroffenen Variablen im Nachhinein zu validieren. Hierzu ist das Attribut `validate` mit "yes" zu belegen. Das Verhalten einer solchen `<assign>`-Aktivität entspricht der sequentiellen Ausführung einer `<assign>`-Aktivität ohne Angabe des `validate`-Attributs (bzw. eines mit "no" belegten `validate`-Attributs) und einem anschließenden `<validate>` der in den `<to>`-Knoten spezifizierten Variablen.

Beispiel: `<assign>`-Aktivität mit `validate="yes"`

```

<assign validate="yes">
    <copy>
        <from variable="zähler"/>
        <to variable="anfrageNr"/>
    </copy>
</assign>

```

```
    </copy>
</assign>
```

Impliziert folgenden BPEL-Code mit <assign>-Aktivität ohne validate-Attribut:

```
<sequence>
  <assign>
    <copy>
      <from variable="zähler"/>
      <to variable="anfrageNr"/>
    </copy>
  </assign>
  <validate variables="anfrageNr"/>
</sequence>
```

3.3.3 Sonstige Basisaktivitäten

Die Aktivität <wait> weist in WS-BPEL 2.0 eine leicht veränderte Syntax auf. Die Zeitspanne, die der Prozess warten soll, wird nicht mehr über ein Attribut spezifiziert, sondern in einem separaten Kindknoten der <wait>-Aktivität beschrieben.

Beispiel: <wait>-Aktivität mit <until>-Bedingung

```
<wait>
  <until>'2006-03-30T18:00+01:00'</until>
</wait>
```

Als Ergänzung zur Aktivität <throw>, gibt es in WS-BPEL 2.0 die Aktivität <rethrow>. Manchmal ist es erforderlich, einen Fehler weiterzureichen, z.B. wenn dieser nur unzureichend behandelt werden kann. Die <throw>-Aktivität benötigt jedoch die Attribute faultName und (optional) faultValue. So kann z.B. ein <catchAll>-Zweig eines Fault Handler nie den zu behandelnden Fehler mittels <throw> weiterreichen, da er keinen Zugriff auf dessen faultName bzw. faultValue hat. Deshalb kann jeder Fault Handler nun mittels der Aktivität <rethrow> den ursprünglichen Fehler erneut werfen. Eventuell vorgenommene Änderungen an den Fehlerdaten werden dabei jedoch verworfen.

Die Syntax der <rethrow>-Aktivität sieht wie folgt aus:

```
<rethrow standard-attributes>
  standard-elements
</rethrow>
```

Die Aktivität <terminate> wurde in <exit> umbenannt.

3.4 Strukturierte Aktivitäten

3.4.1 Auswahl von Alternativen

Die aus BPEL4WS 1.1 bekannte Aktivität `<switch>` wurde in `<if>` umbenannt. Die `<case>`- und `<otherwise>`-Zweige werden nun entsprechend als `<elseif>` und `<else>` bezeichnet. Die Bedingungen werden nicht mehr in Attributen, sondern in separaten `<condition>`-Elementen beschrieben.

Beispiel: Eine `<if>`-Aktivität in WS-BPEL 2.0

```
<if>
  <condition>
    bpws:getVariableProperty('bestellung','zahlungsart')='Kreditkarte'
  </condition>
  <flow>
    <!-- Zahlung per Kreditkarte abwickeln -->
  </flow>
  <elseif>
    <condition>
      bpws:getVariableProperty('bestellung','zahlungsart')='Bankeinzug'
    </condition>
    <flow>
      <!-- Zahlung per Bankeinzug abwickeln -->
    </flow>
  </elseif>
  <else>
    <throw faultName="NichtUnterstützteZahlungsart"/>
  </else>
</if>
```

Zum Vergleich, die entsprechende `<switch>`-Aktivität in BPEL4WS 1.1:

```
<switch>
  <case condition=
    "bpws:getVariableProperty(bestellung,zahlungsart)='Kreditkarte'">
    <flow>
      <!-- Zahlung per Kreditkarte abwickeln -->
    </flow>
  </case>
  <case condition=
    "bpws:getVariableProperty(bestellung,zahlungsart)='Bankeinzug'">
    <flow>
```

```

        <!-- Zahlung per Bankeinzug abwickeln -->
    </flow>
</case>
<otherwise>
    <throw faultName="NichtUnterstützteZahlungsart"/>
</otherwise>
</switch>

```

Die `<if>`- bzw. `<switch>`-Aktivität aus obigem Beispiel könnte z.B. Teil eines Zahlungsvorgangs für eine Bestellung in einem Web-Shop sein. Zuerst wird überprüft, ob es sich um eine Zahlung mit Kreditkarte handelt. Alternativ kann auch per Bankeinzug bezahlt werden. Weitere Zahlungsarten werden in diesem Beispiel nicht unterstützt und produzieren einen Fehler. Der Ablauf der beiden möglichen Zahlungsvorgänge wird hier aus Platzgründen nur über einen Kommentar angedeutet. Semantisch sind die `<switch>`-Aktivität aus BPEL4WS 1.1 und die `<if>`-Aktivität aus WS-BPEL 2.0 äquivalent.

Was im Abschnitt 3.3.1 zu den Kommunikationsaktivitäten erläutert wurde, lässt sich in weiten Teilen auch auf die Aktivität `<pick>` übertragen:

- Das `portType`-Attribut ist in WS-BPEL 2.0 optional.
- `<onMessage>`-Zweige können über das optionale Attribut `messageExchange` mit einer `<reply>`-Aktivität assoziiert werden.
- Alternativ zur Verwendung des `variable`-Attributs können `<onMessage>`-Zweige `<fromPart>`-Elemente verwenden.

Die Bedingung eines `<onAlarm>`-Zweigs wird nicht mehr in einem Attribut, sondern in einem separaten Kindknoten beschrieben.

3.4.2 Schleifen

Die Aktivität `<while>` wurde nur syntaktisch leicht verändert. Auch hier wird die boolesche Bedingung nicht mehr über ein Attribut, sondern über ein separates `<condition>`-Element spezifiziert (siehe 3.3.3).

Die neue Aktivität `<repeatUntil>` bietet, ebenso wie `<while>`, die Möglichkeit zur wiederholten Ausführung einer beliebigen Aktivität. Allerdings wird im Gegensatz zu `<while>` die boolesche Bedingung erst nach der Ausführung des Schleifenkörpers überprüft. `<repeatUntil>` bietet sich also insbesondere dann an, wenn die Aktivität innerhalb der Schleife mindestens einmal ausgeführt werden soll.

Die Syntax der `<repeatUntil>`-Aktivität sieht wie folgt aus:

```
<repeatUntil standard-attributes>
  standard-elements
  activity
  <condition expressionLanguage="anyURI"?>
    bool-expr
  </condition>
</repeatUntil>
```

WS-BPEL 2.0 bietet mit der neuen `<forEach>`-Aktivität ein weiteres Konstrukt zur wiederholten Ausführung einer Scope-Aktivität.

Die Syntax der `<forEach>`-Aktivität ist:

```
<forEach counterName="ncname" parallel="yes|no" standard-attributes>
  standard-elements
  <iterator>
    <startCounterValue expressionLanguage="anyURI">
      ...
    </startCounterValue>
    <finalCounterValue expressionLanguage="anyURI">
      ...
    </finalCounterValue>
  </iterator>
  <completionCondition>
    <branches expressionLanguage="URI"?
      countCompletedBranchesOnly="yes|no"?>
      an-integer-expression
    </branches>
  </completionCondition?>
  scope-activity
</forEach>
```

Beim Start der `<forEach>`-Aktivität werden die Ausdrücke in den `<startCounterValue>`- und `<finalCounterValue>`-Elementen einmalig ausgewertet und bleiben dann über die gesamte Laufzeit der Aktivität konstant. Die dort spezifizierten Ausdrücke müssen ein *xs:unsignedint* als Ergebnistyp haben, sonst wird der Fehler *bpws:forEachCounterError* geworfen. Sollte der Wert von `<startCounterValue>` größer sein, als der von `<finalCounterValue>`, wird keine Iteration durchgeführt.

Über das Attribut `parallel` kann gesteuert werden, ob der Schleifenkörper sequentiell oder parallel abgearbeitet wird.

Im Fall von `parallel="no"` wird die Schleife sequentiell abgearbeitet. Die eingebettete Scope-Aktivität wird $N+1$ Mal (wobei N die Differenz von `<finalCounterValue>` und `<startCounterValue>` ist) nacheinander ausgeführt. Für jeden Durchlauf wird eine Variable vom Typ `xs:unsignedint` kreiert, die dann mit dem aktuellen Zählerwert initialisiert wird. Der Bezeichner dieser Variable wird über das Attribut `counterName` der `<forEach>`-Aktivität festgelegt. Im ersten Durchlauf wird sie auf den Wert von `<startCounterValue>` initialisiert und bei jedem weiteren Durchlauf inkrementiert. Im letzten Durchlauf besitzt die Zählervariable den Wert von `<finalCounterValue>`. Die Zählervariable ist lokal für den eingebetteten Scope, kann also von diesem gelesen und geschrieben werden. Nach einer Abarbeitung des Schleifenkörpers werden jedoch alle Änderungen an dieser Variable verworfen.

Im Fall von `parallel="yes"` wird die Schleife parallel abgearbeitet. Es wird eine implizite Flow-Aktivität generiert, die $N+1$ Instanzen der eingebetteten Scope-Aktivität enthält. Für jede dieser Instanzen wird eine Zählervariable (wie im sequentiellen Fall) kreiert, die eindeutig mit einem Wert aus dem Intervall `<startCounterValue>` bis `<finalCounterValue>` initialisiert wird.

Optional kann eine `<forEach>`-Aktivität ein `<completionCondition>`-Element enthalten. Wird die in diesem Element spezifizierte Bedingung während der Abarbeitung erfüllt, so kann die `<forEach>`-Aktivität vorzeitig beendet werden. Das `<completionCondition>`-Element enthält das Element `<branches>`, in dem ein Integer-Ausdruck angegeben wird. Dieser wird beim Start der `<forEach>`-Aktivität einmalig ausgewertet. Ist der Wert des Ausdrucks größer als die Anzahl der Schleifendurchläufe (bzw. parallelen Instanzen), wird der Fehler `bpws:invalidBranchCondition` geworfen. Nach jeder Abarbeitung eines eingebetteten Scope wird die Anzahl der abgearbeiteten Scopes mit dem Wert dieses Ausdrucks verglichen. Ist die Anzahl der abgearbeiteten Scopes größer oder gleich dem Wert des Ausdrucks, gilt die Bedingung als erfüllt. Es lassen sich also Bedingungen nach dem Muster "mindestens M aus N "⁶ definieren. Über das optionale Attribut `countCompletedBranchesOnly` des `<branches>`-Elements kann festgelegt werden, ob dabei jede Abarbeitung oder nur jede erfolgreiche (fehlerfreie) Abarbeitung eines eingebetteten Scope gezählt wird.

Die `<completionCondition>` wird immer dann ausgewertet, wenn ein eingebetteter Scope beendet wurde. Wenn feststeht, dass die Bedingung nicht mehr erfüllt werden kann, wird der Fehler `bpws:completionConditionFailure` geworfen. Ist die Bedingung erfüllt, wird die `<forEach>`-Aktivität erfolgreich beendet. Im sequentiellen Fall werden dann alle noch ausstehenden Scope-Aktivitäten nicht mehr ausgeführt. Im parallelen Fall werden alle noch aktiven Scope-Aktivitäten beendet.

⁶"Mindestens" deshalb, da es sich im parallelen Fall möglicherweise um mehr als N handeln kann.

3.5 Link Semantik

Die Spezifikation von WS-BPEL 2.0 beschreibt die Link Semantik genauer, als dies in BPEL4WS 1.1 der Fall ist. Neu sind u.a. folgende explizit gestellte Restriktionen:

- Je zwei Aktivitäten dürfen nur durch maximal einen Link verbunden sein.
- Links, deren Quelle im XML-Teilbaum der dazugehörigen Zielaktivität definiert ist, sind nicht gestattet.
- Für den Fall, dass zwei ineinander geschachtelte `<flow>`-Aktivitäten je einen Link mit dem gleichen Namen definieren, wird die Namensauflösung von innen nach außen betrieben, d.h. weiter außen stehende Links werden von weiter innen stehenden Links mit gleichem Namen überladen.
- Ausgehende Links von Fault und Termination Handler (spezieller Handler für die erzwungene Terminierung von Aktivitäten, siehe 3.9), müssen auf "negative" gesetzt werden, sobald feststeht, dass diese nicht abgearbeitet werden.

`<source>`- und `<target>`-Elemente können in WS-BPEL 2.0 in einem `<sources>`- bzw. `<targets>`-Element gekapselt werden. Die `joinCondition` bzw. `transitionCondition` wird nicht mehr über ein Attribut, sondern in einem separaten `<joinCondition>`- bzw. `<transitionCondition>`-Element definiert.

Beispiel: `<sources>`-Element mit `<transitionCondition>`

```
<sources>
  <source linkName="kontaktdaten-erfragen">
    <transitionCondition>
      $kunde.status='Neukunde'
    </transitionCondition>
  </source>
  <source linkName="versand-vorbereiten">
    <transitionCondition>
      $kunde.status!='Neukunde'
    </transitionCondition>
  </source>
</sources>
```

Das obige BPEL-Fragment könnte z.B. Teil eines Web Service sein, der den Bestellvorgang innerhalb eines Web-Shops verwaltet. Nachdem der Kunde seine Produktauswahl getroffen hat und eine Bestellung auslösen möchte, kann in Abhängigkeit seines Status entweder direkt der Versand der Ware veranlasst werden (der Kunde ist bereits beim Web-Shop registriert) oder es

müssen zuerst die Kontaktdaten des Kunden erfragt werden (es handelt sich um einen Neukunden). Durch die Auswertung der `<transitionCondition>` wird der entsprechende Link auf “positive” bzw. “negative” gesetzt und damit das weitere Verhalten des Prozesses bestimmt.

Die `<transitionCondition>` wird ausgewertet, nachdem die Aktivität, innerhalb der das entsprechende `<source>`-Element definiert ist, abgearbeitet wurde. Tritt während der Auswertung der Bedingung ein Fehler auf, hat dieser keinen Einfluss auf den Beendigungsstatus⁷ der Aktivität und wird von dem Scope behandelt, der das `<source>`-Element umgibt. Alle weiteren ausgehenden Links der Aktivität (und deren `<transitionCondition>`) werden in diesem Fall nicht mehr ausgewertet. Der Status dieser Links ist also “negative”.

Die Semantik der Dead-Path-Elimination (DPE) wird durch WS-BPEL 2.0 nicht verändert.

3.6 Scope

Die Spezifikation von WS-BPEL 2.0 beschreibt das Verhalten von Scopes für den Fall der Initialisierung und Beendigung.

Die Abarbeitung eines Prozesses bzw. Scopes beginnt mit der Initialisierung. Dabei werden die Fault und Termination Handler, Partner Links, Correlation Sets und die Variablen instanziiert und initialisiert. Die Partner Links müssen erstellt werden, bevor die im gleichen Scope definierten Variablen initialisiert werden können. Die Scope-Initialisierung verläuft nach dem Motto: alles oder nichts. Entweder alles wird erfolgreich initialisiert oder es wird der Fehler *bpws:scopeInitializationFailure* geworfen, der vom umgebenden Scope behandelt werden muss. Im Falle eines solchen Fehlers auf Prozessebene gilt der gesamte Prozess als “faulted”. Nachdem die Initialisierung abgeschlossen ist, werden die erste innere Aktivität und die Event Handler des Scope parallel instanziiert. Eine Ausnahme dazu stellen Scopes dar, die eine *initial start activity* enthalten (d.h. die Instanziierung einer weiteren Prozessinstanz bewirken). In diesem Fall muss zuerst die *initial start activity* beendet worden sein, bevor die Event Handler instanziiert werden können.

Wird ein Scope vollständig abgearbeitet, müssen alle Interaktionen mit Web Services, die abhängig von innerhalb des Scope definierten Partner Links und `messageExchange`-Definitionen sind, beendet sein. Bleiben Empfangsaktivitäten offen, die sich auf Partner Links oder `messageExchange`-Definitionen des Scope beziehen, kann der Fehler *bpws:missingReply* geworfen werden.⁸

Ein Scope (bzw. Prozess) kann in WS-BPEL 2.0 das `exitOnStandardFault`-Attribut besitzen. Falls der Wert dieses Attributs auf “yes” gesetzt ist, muss der Prozess sofort beendet werden (äquivalent zum Erreichen der Aktivität `<exit>`), wenn ein anderer WS-BPEL Standardfehler als *bpws:joinFailure* auftritt. Ist Wert des Attributs hingegen “no”, so kann der Prozess den Standardfehler mittels eines Fault Handler behandeln. Der Standardwert des `exitOnStandardFault`-Attributs ist “no”. Wird dieses Attribut von einer Scope-Aktivität nicht spezifiziert, so wird der Wert vom umgebenden Scope bzw. Prozess geerbt.

⁷Eine Aktivität wird entweder vollständig (fehlerfrei) oder unvollständig (fehlerhaft bzw. vorzeitig) beendet.

⁸Siehe WS-BPEL 2.0 Specification, 12. Scopes.

3.6.1 Isolierte Scopes

Serialisierbare Scopes werden in WS-BPEL 2.0 als isolierte Scopes bezeichnet. Das Attribut `variableAccessSerializable` heißt nun entsprechend `isolated`. Neu ist auch, dass ein als isoliert markierter Scope weitere Scopes einbetten kann, die nicht als isoliert markiert sind.⁹ Der Zugriff auf gemeinsam genutzte Variablen durch die so eingebetteten Scopes wird dann über den isolierten Scope kontrolliert.

Beachte, dass die Isolation eines Variablenzugriffs nicht zu einem internen Deadlock in einem BPEL-Prozess führen kann. Der Grund dafür liegt darin, dass ein isolierter Scope nicht eher gestartet wird, als bis er exklusiven Zugriff auf alle vom ihm benötigten nicht-lokalen Variablen hat.

3.7 Event Handler

Event Handler führen in WS-BPEL 2.0 grundsätzlich nur noch Scope-Aktivitäten aus, um eine saubere Scope-Snapshot- und Kompensationssemantik zu gewährleisten. Dies ist natürlich keine Einschränkung, da die Scope-Aktivität beliebige weitere Aktivitäten enthalten kann. Tritt während der Abarbeitung eines Event Handler ein Fehler auf, wird dieser also zuerst vom Fault Handler der eingebetteten Scope-Aktivität behandelt bzw. von ihm an den umgebenden Scope weiter gereicht.

Sowohl von `<onEvent>`- als auch von `<onAlarm>`-Event-Handler können zu einem Zeitpunkt mehrere aktive Instanzen existieren. Jede Instanz erhält daher eine private Kopie der im eingebetteten Scope deklarierten Daten und Ressourcen (inkl. Links und Partner Links), um Zugriffskonflikte zu verhindern.

3.7.1 Message Events

Das `<onMessage>`-Element des Event Handler wurde in `<onEvent>` umbenannt. Was im Abschnitt 3.3.1 zu den Kommunikationsaktivitäten erläutert wurde, lässt sich in weiten Teilen auch auf die `<onEvent>`-Elemente übertragen:

- Das `portType`-Attribut ist in WS-BPEL 2.0 optional.
- `<onEvent>`-Elemente können über das optionale Attribut `messageExchange` mit einer `<reply>`-Aktivität assoziiert werden.
- Alternativ zur Verwendung des `variable`-Attributs können `<fromPart>`-Elemente genutzt werden.

⁹In BPEL4WS 1.1 darf ein serialisierbarer Scope keinerlei weitere Scopes einbetten.

Es besteht nun die Möglichkeit, innerhalb eines Event-Handler-Zweigs lokal Correlation Sets zu deklarieren, die dann bei Empfang der Nachrichten verwendet werden können.

Sollte zu einem Zeitpunkt mehr als eine Empfangsaktivität für das gleiche Tupel aus `partnerLink`, `portType`, `operation` und Correlation Set aktiviert sein, so wird der Fehler *bpws:conflictingReceive* geworfen.

3.7.2 Alarm Events

Das `<onAlarm>`-Element beschreibt ein zeitgesteuertes Ereignis. Im Gegensatz zu BPEL4WS 1.1 wird der Eintrittszeitpunkt des Ereignisses in WS-BPEL 2.0 nicht mehr über ein Attribut, sondern in einem separaten `<for>`- bzw. `<until>`-Element spezifiziert. Mit dem optionalen `<repeatEvery>`-Element kann nach Ablauf der Zeitdauer wiederholt die Stoppuhr gestartet werden, solange der Scope, in dem der Event Handler definiert wurde, aktiv ist. Das `<repeatEvery>`-Element kann sowohl allein, als auch in Kombination mit entweder einem `<for>`- oder einem `<until>`-Element stehen. Steht es allein, so wird die Uhr erstmalig mit dem Start des den Event Handler umgebenden Scope gestartet. Steht es in Kombination mit einem `<for>`- oder `<until>`-Element, so wird die Uhr nicht eher gestartet, als die durch das `<for>`- oder `<until>`-Element definierte Zeitdauer abgelaufen ist. Danach wird das Ereignis jeweils nach Ablauf der im `<repeatEvery>`-Element spezifizierten Zeitdauer signalisiert.

3.8 Fault Handler

Die WS-BPEL 2.0 Spezifikation beschreibt die optional zu einem Fehlernamen assoziierten Fehlerdaten weitaus konkreter, als dies bisher der Fall war.

Fehlerdaten sind WSDL-Nachrichtentypen oder XML-Schema-Elemente. Jeder `<catch>`-Zweig, der einen `faultName` spezifiziert kann nur Fehler eines einzigen Typs behandeln. Wenn die zu behandelnden Fehlerdaten von einem WSDL-Nachrichtentyp sind, dann muss dieser mittels des (neuen optionalen) `faultMessageType`-Attributs spezifiziert sein. Wenn die Fehlerdaten eine XML-Element-Definition sind, dann muss diese mittels des (neuen optionalen) `faultElement`-Attributs spezifiziert sein.

Da das Attribut `faultName` optional ist, kann es vorkommen, dass eine spezifizierte `faultVariable` keinem konkreten Typ zugeordnet werden kann. Um dies zu verhindern, muss bei der Spezifikation des `faultVariable`-Attributs stets auch entweder ein `faultMessageType`- oder ein `faultElement`-Attribut mit angegeben werden. Zudem dürfen sowohl das `faultMessageType`- als auch das `faultElement`-Attribut nie ohne ein begleitendes `faultVariable`-Attribut angegeben werden.

Für die Behandlung eines Fehlers können mehrere Fault Handler zur Auswahl stehen. Die Regeln, nach denen ein Fault Handler zur Fehlerbehandlung ausgewählt wird, wurden in Anbetracht der erweiterten Möglichkeiten von WS-BPEL 2.0 entsprechend angepasst.¹⁰

¹⁰Siehe WS-BPEL 2.0 Specification, 12.4 Fault Handlers.

3.9 Termination Handler

Scopes besitzen die Möglichkeit, auf den Ablauf einer erzwungenen Terminierung Einfluss zu nehmen. Was in BPEL4WS 1.1 als Forced-Termination-Zweig des Fault Handler definiert wurde, wird in WS-BPEL 2.0 innerhalb des Scope als Termination Handler definiert. Dieser kann die selben Aktivitäten verwenden, die auch in einem Fault Handler verwendet werden können. Es wird also nicht mehr der Fehler *bpws:forcedTermination* geworfen, auf den dann im Fault Handler reagiert werden kann, sondern im Falle einer erzwungenen Terminierung wird nach der Beendigung aller laufenden Aktivitäten des Scope der Termination Handler ausgeführt. Ist kein Termination Handler definiert, wird ein Standard-Termination-Handler aktiviert. Dieser kompensiert alle erfolgreich beendeten eingebetteten Scopes in der standardmäßigen Kompensationsreihenfolge (siehe 3.10.1). Er verhält sich also wie der Standard-Fault-Handler.

Die Syntax des Termination Handler ist:

```
<terminationHandler>?  
    ...  
</terminationHandler>
```

3.10 Compensation Handler

Der aktuelle Zustand des Prozesses setzt sich aus den aktuellen Zuständen aller gestarteten Scopes zusammen. Dies beinhaltet Scopes die erfolgreich abgearbeitet wurden, deren Compensation Handler aber noch nicht aufgerufen wurde. Der aktuelle Zustand eines erfolgreich abgearbeiteten Scope entspricht dessen Zustand zum Zeitpunkt der Beendigung der Abarbeitung. Ein Scope kann, möglicherweise als Teil eines Schleifendurchlaufs, mehrfach ausgeführt worden sein. In diesem Fall enthält der aktuelle Zustand des Prozesses den Zustand einer jeder erfolgreichen (nicht kompensierten) Ausführung eines solchen Scope. Dieser gesicherte Zustand eines erfolgreich abgearbeiteten unkompensierten Scope wird als Scope-Snapshot bezeichnet.

Compensation Handler nutzen in WS-BPEL 2.0 immer den aktuellen Zustand des Scope, d.h. den Zustand des Scope, in dem sie definiert wurden und allen darin eingebetteten Scopes. Dies umfasst die dort deklarierten Variablen, Partner Links und Correlation Sets. Sie sind in der Lage, die Werte all dieser Variablen zu lesen und zu setzen. Andere Teile des Prozesses können die Veränderung an gemeinsam genutzten Variablen sehen, genauso wie die Compensation Handler Veränderungen sehen können, die durch andere Teile des Prozesses an gemeinsam genutzten Variablen vorgenommen werden. Dies gilt auch für den Fall einer parallelen Abarbeitung. Um Konflikte im Falle von erwarteter gleichzeitiger Abarbeitung zu verhindern, müssen Compensation Handler isolierte Scopes (siehe 3.6.1) verwenden, wenn sie in den Zustand eines eingebetteten Scope eingreifen.

Wie auch in BPEL4WS 1.1 steht ein Compensation Handler nur für Scopes zur Verfügung, die ordnungsgemäß abgearbeitet wurden. Wird ein Compensation Handler wiederholt aufgerufen,

wird dies in WS-BPEL 2.0 ignoriert.¹¹ Dabei spielt es keine Rolle, ob es sich um einen aktivierten oder deaktivierten Compensation Handler handelt.

Die aus BPEL4WS 1.1 bekannte Aktivität `<compensate>` wurde in zwei Aktivitäten zur Kompensation aufgeteilt: `<compensate>`, zur Kompensation aller eingebetteten Scopes in der standardmäßigen Kompensationsreihenfolge (siehe 3.10.1) und `<compensateScope>` zur Kompensation eines speziellen Scope. Ist der zu kompensierende Scope in eine Schleife oder einen Event Handler eingebettet, müssen alle Iterationen der Schleife bzw. alle verarbeiteten Ereignisse des Event Handler kompensiert werden. Die für eine solche Aktivität im Laufe der Abarbeitung aktivierten Compensation Handler werden dabei als eine Einheit betrachtet, die als *Compensation Handler Instance Group* bezeichnet wird. Im Falle eines Aufrufs von `<compensate>`, umfasst diese Compensation Handler Instance Group alle Compensation-Handler-Instanzen der erfolgreich abgearbeiteten Scopes, die in der Aktivität eingebettet sind. Im Falle eines Aufrufs von `<compensateScope>`, jedoch nur die Compensation-Handler-Instanzen des spezifizierten Ziel-Scope. Sollte während der Abarbeitung einer dieser Compensation-Handler-Instanzen ein unbehandelter Fehler auftreten, wird die gesamte Compensation Handler Instance Group beendet und alle enthaltenen Compensation Handler werden deaktiviert.

Die Syntax der `<compensate>`-Aktivität sieht wie folgt aus:

```
<compensate standard-attributes>
  standard-elements
</compensate>
```

Die Syntax der `<compensateScope>`-Aktivität ist:

```
<compensateScope target="ncname"? standard-attributes>
  standard-elements
</compensateScope>
```

3.10.1 Standardmäßige Kompensationsreihenfolge

Die Spezifikation von WS-BPEL 2.0 beschreibt zwei Regeln, die für eine standardmäßige Kompensationsreihenfolge gelten müssen. Um diese Regeln zu konkretisieren, werden formal einige Begriffe definiert.¹² Wir wollen uns jedoch hier nur auf eine informelle Wiedergabe dieser Regeln beschränken:

1. Die Kompensation muss sich nach der Reihenfolge der Abarbeitung der zu kompensierenden Scopes richten, soweit diese durch die Prozessdefinition vorgegeben ist.

¹¹In BPEL4WS 1.1 produziert der wiederholte Aufruf eines aktivierten Compensation Handler den Fehler *bpws:repeatedCompensation*.

¹²Siehe WS-BPEL 2.0 Specification, 12.4.2 Default Compensation Order.

2. Prozesse, in denen es zu Zyklen durch Links kommen kann, sind nicht gestattet, d.h. wenn von einem Scope A ein Link in einen anderen Scope B führt, darf von Scope B kein Link in den Scope A zurückführen.

Für den Fall von nebenläufig abgearbeiteten Scopes, die keine Kontrollflussabhängigkeiten durch Links aufweisen, wird durch diese Regeln keine konkrete Kompensationsreihenfolge festgelegt. Sie können also sowohl nebenläufig, als auch in strikter umgekehrter Reihenfolge ihrer Abarbeitung nach einander kompensiert werden.

3.10.2 Kompensation und isolierte Scopes

Compensation Handler eines isolierten Scope werden, anders als entsprechende Fault Handler, nicht innerhalb dessen isolierter Umgebung ausgeführt. Auch können diese Compensation Handler nicht selbst isolierte Scopes verwenden, da isolierte Scopes nicht in einander geschachtelt werden dürfen. Dies wirft jedoch Fragen in Bezug auf die Isolationssemantik der Compensation Handler von in isolierten Scopes eingebetteten Scopes auf. Wie sieht deren Umgebung im Falle eines Aufrufs durch einen Fault Handler (ausgeführt innerhalb der isolierten Umgebung des umgebenden Scope) bzw. im Falle eines Aufrufs durch einen Compensation Handler (ausgeführt außerhalb der isolierten Umgebung des umgebenden Scope) aus?

Um konsistentes Verhalten sicher zu stellen, verlangt die Spezifikation von WS-BPEL 2.0 das Compensation Handler innerhalb eines isolierten Scope daher selbst implizit isoliertes Verhalten zeigen, obwohl dies eine separate isolierte Umgebung erfordert.

3.11 Erweiterbarkeit

WS-BPEL 2.0 Prozesse lassen sich in vielerlei Hinsicht erweitern. Diese Erweiterung kann von neuen Attributen, Elementen und Aktivitäten über erweiterte Datenmanipulationen (siehe 3.3.2 `<extendableAssign>`) bis hin zu Restriktionen bzw. Erweiterungen des Laufzeitverhaltens eines Prozesses reichen. Auf Prozessebene (d.h. direkt innerhalb der `<process>`-Aktivität) können die XML-Namensräume der neuen Elemente mit Hilfe von `<extension>`-Elementen angegeben werden. Die Syntax dafür sieht wie folgt aus:

```
<extensions>?  
  <extension namespace="anyURI" mustUnderstand="yes|no"/> *  
</extensions>
```

Werden mit `mustUnderstand="yes"` deklarierte Erweiterungen von der Laufzeitumgebung nicht unterstützt, muss diese den Prozess zurückweisen.

Damit die Semantik der Erweiterungselemente angewandt wird, müssen diese in der Prozessdefinition unter Angabe ihres Namensraums spezifiziert sein, z.B.:

```
<scope>
  <sequence>
    <invoke operation="operation1"
           foo:invokeProperty="eigenschaft1" ... />
    <invoke operation="operation2" ... />
    <invoke operation="operation3"
           foo:invokeProperty="eigenschaft2" ... />
  </sequence>
</scope>
```

Die Erweiterungen gelten nur in dem XML-Teilbaum ihres Eltern-Elements, d.h. im obigen Beispiel wird `foo:invokeProperty` nur auf die `<invoke>`-Aktivitäten für `operation1` und `operation3` angewandt.

Mit der neuen Basisaktivität `<extensionActivity>`, bietet WS-BPEL 2.0 die Möglichkeit, die Sprache um neue Aktivitäten zu erweitern. Innerhalb einer `<extensionActivity>` kann immer nur genau eine neue Aktivität stehen. Diese muss die WS-BPEL 2.0 Standardattribute und Standardelemente unterstützen. Wird die Aktivität von einer BPEL-Laufzeitumgebung nicht erkannt und nicht in einer Erweiterungsdeklaration über das Attribut `mustUnderstand="yes"` als zwingend erforderlich deklariert, wird sie als `<empty>`-Aktivität behandelt. In jedem Fall werden jedoch die Standardattribute und Standardelemente entsprechend der WS-BPEL 2.0 Spezifikation verarbeitet.

Die Syntax der `<extensionActivity>`-Aktivität (für eine neue Aktivität `<myNewActivity>`) sieht wie folgt aus:

```
<extensionActivity>
  <myNewActivity standard-attributes>
    standard-elements
  </myNewActivity>
</extensionActivity>
```

3.12 Abstrakte Prozesse

Die Möglichkeiten zur Definition abstrakter Prozesse wurden in WS-BPEL 2.0 deutlich erweitert. Die Spezifikation legt eine *Common Base* als syntaktische Basis abstrakter Prozesse fest und definiert, wie mit Hilfe von Profilen die Semantik dieser Prozesse beschrieben werden kann. Durch die Definition neuer Profile lässt sich das Einsatzfeld abstrakter Prozesse beliebig erweitern.

3.12.1 Common Base

Die Common Base beschreibt die zulässige Syntax für abstrakte Prozesse wie folgt:

- Der Prozess muss das `abstractProcessProfile`-Attribut besitzen, das auf eine existierende Profildefinition verweisen muss.
- Alle Sprachkonstrukte ausführbarer Prozesse sind erlaubt.
- Bestimmte Elemente (wie Attribute, Aktivitäten, etc.) dürfen explizit (durch Platzhalter) oder implizit (durch Auslassung) versteckt sein.
- Abstrakte Prozesse müssen der statischen Validierung von WS-BPEL 2.0 genügen.
- Die für ausführbare Prozesse obligatorische "createInstance"-Aktivität kann ausgelassen werden.

Details der Ausführung dürfen in abstrakten Prozessen durch Platzhalter ersetzt werden. Diese Platzhalter können für Ausdrücke, Aktivitäten und Attribute stehen. Ein vom abstrakten Prozess abgeleiteter ausführbarer Prozess muss diese Platzhalter durch entsprechende ausführbare BPEL-Konstrukte ersetzen. Dabei können verschiedene ausführbare Ableitungen unterschiedliche Ersetzungen für die Platzhalter vornehmen.

Die Aktivität `<opaqueActivity>` kann in abstrakten Prozessen als Platzhalter für beliebige Aktivitäten verwendet werden. Sie kann zum Beispiel in einem abgeleiteten ausführbaren Prozess als Erweiterungspunkt dienen.

Die Syntax der `<opaqueActivity>`-Aktivität ist:

```
<opaqueActivity standard-attributes>
  standard-elements
</opaqueActivity>
```

Die für die Platzhalter-Aktivität spezifizierten Standardattribute und Standardelemente müssen dabei von jeder ausführbaren Ableitung übernommen werden.

Platzhalter für Ausdrücke und Zuweisungen können über das nur für abstrakte Prozesse zulässige Attribut `opaque="yes"` definiert werden. So können z.B. Nichtdeterminismus für Auswahl von Alternativen und Verwendung von unternehmensinternen Daten ausgedrückt werden.

Beispiele für Ausdrücke und Zuweisungen mit Attribut `opaque="yes"`:

```
<transitionCondition expressionLanguage="anyURI"? opaque="yes"/>
<from opaque="yes"/>
```

Um einen Platzhalter für ein bestimmtes Attribut einer Aktivität zu definieren, kann das entsprechende Attribut in einem abstrakten Prozess mit dem Wert `##BPELOpaque` belegt werden.

Elemente eines Prozesses, die ohnehin syntaktisch erforderlich sind und keinen Standardwert besitzen, können durch eine Auslassung implizit versteckt werden. Dies ist äquivalent zur Definition eines Platzhalters für das entsprechende Element.

Zu jedem abstrakten Prozess muss es mindestens eine ausführbare Basisvervollständigung geben, um die statische Validierung über einen simplen XML-Schema-Check hinaus zu gewährleisten. Diese Bedingung beschränkt die Anzahl absurder Konstruktionen in abstrakten Prozessen, die rein vom XML-Schema her zulässig wären.

Für eine ausführbare Basisvervollständigung sind dabei die folgenden syntaktischen Transformationen erlaubt:

- Änderung des verwendeten Namensraums in den für ausführbare Prozesse und Entfernen des `abstractProcessProfile`-Attributs.
- Ersetzung der verwendeten Platzhalter durch entsprechende ausführbare Elemente.
- Hinzufügen beliebiger BPEL-XML-Elemente an beliebigen Stellen innerhalb des Prozesses, unter Wahrung der statischen Validierbarkeit.

3.12.2 Profile

Um die Interpretation abstrakter Prozesse zu ermöglichen, reicht die Definition der Common Base nicht aus. Mit Hilfe von Profilen können Klassen abstrakter Prozesse mit gleicher Semantik definiert werden. Ein Profil beschreibt dabei die zulässigen Möglichkeiten der Vervollständigung seiner abstrakten Prozesse. So kann zum Beispiel durch ein Profil nur das Ersetzen vorhandener Platzhalter, nicht jedoch das Hinzufügen neuer Aktivitäten erlaubt werden oder das Hinzufügen neuer Aktivitäten insofern beschränkt werden, als das diese nur Blätter im XML-Baum bilden dürfen.

Die Spezifikation von WS-BPEL 2.0 enthält bereits die beiden Profile “Abstract Process Profile for Observable Behavior” und “Abstract Process Profile for Templates”. Das “Abstract Process Profile for Observable Behavior” findet im Bereich der Beschreibung von beobachtbarem Verhalten eines Geschäftspartners im Kontext von Web Services Anwendung.¹³ Das “Abstract Process Profile for Templates” kann eingesetzt werden, um Schablonen für Prozesse zu definieren, deren Details der Ausführung (wie z.B. Endpoint-Referenzen) später ergänzt werden können.¹⁴

Beispiele für abstrakte Prozesse finden sich in der Spezifikation von WS-BPEL 2.0 unter “14.1. Shipping Service: Observable Behavior Profile Abstract Process” für das Abstract Process Profile for Observable Behavior und unter “13.4.5. Extensions and document usage” für das Abstract Process Profile for Templates.

¹³Siehe WS-BPEL 2.0 Specification, 13.3 Abstract Process Profile for Observable Behavior.

¹⁴Siehe WS-BPEL 2.0 Specification, 13.4 Abstract Process Profile for Templates.

4 Petrinetz-Semantik für WS-BPEL 2.0

Die von Stahl [Sta05] entwickelte Petrinetz-Semantik für BPEL4WS 1.1 soll als Grundlage für eine Petrinetz-Semantik für WS-BPEL 2.0 dienen. Viele der in diesem Dokument vorgestellten Änderungen von WS-BPEL 2.0 sind für die Petrinetz-Semantik irrelevant und haben daher keine Auswirkung auf die vorhandene Petrinetz-Semantik für BPEL4WS 1.1. Dies liegt zum einen daran, dass die Petrinetz-Semantik unabhängig von der BPEL-Syntax ist und zum anderen an der gewählten Abstraktionsebene. So werden Aspekte der Datenmodellierung und der Fehlertypen bisher gar nicht oder nur in begrenztem Umfang von der Petrinetz-Semantik berücksichtigt.

Die folgenden Abschnitte geben einen Überblick über die für die Petrinetz-Semantik relevanten Änderungen von WS-BPEL 2.0.

4.1 Neue Aktivitäten

Für die in WS-BPEL 2.0 neu spezifizierten Aktivitäten müssen entsprechende Petrinetz-Muster entwickelt werden. Dies betrifft folgende Aktivitäten:

- `<validate>`
- `<repeatUntil>`
- `<forEach>` (sequentiell und parallel)
- `<rethrow>`
- `<compensateScope>`

Die Aktivität `<extensionActivity>` kann dabei nicht berücksichtigt werden, da deren Semantik in Abhängigkeit der gekapselten Aktivität variabel ist.

4.2 Correlation

Das `initiate`-Attribut eines `<correlationSet>` kann nun auch den Wert "join" annehmen. Um dem auch in der Petrinetz-Semantik Rechnung zu tragen, gilt es, für folgende Aktivitäten entsprechend zusätzliche Muster zu entwickeln:

- <receive>
- <reply>
- <invoke> (synchron und asynchron)
- <pick>
- Event Handler (<onEvent>-Element)

4.3 Datenmanipulation

Die <assign>-Aktivität besitzt nun ein optionales Attribut `validate`. Im Falle von `validate="no"` ist das bereits vorhandene <assign>-Muster zu verwenden, für `validate="yes"`, die sequentielle Verknüpfung des vorhandenen <assign>-Musters mit dem neuen <validate>-Muster. Dies kann durch statische Analyse festgestellt und bei der Generierung des Petrinetzes für den gesamten Prozess berücksichtigt werden.

4.4 Scope

In einem Scope, der eine *initial start activity* enthält, muss zuerst die *initial start activity* beendet worden sein, bevor die Event Handler instanziiert werden dürfen. Das vorhandene Scope-Muster ist dahingehend anzupassen, dass es dieses Verhalten modelliert.

Ein Scope kann nun einen Termination Handler besitzen, der den Ablauf einer erzwungenen Termination regeln kann. Das vorhandene Forced-Termination-Muster des Fault Handler muss entsprechend angepasst bzw. in das Scope-Muster integriert werden.

4.5 Event Handler

Das <onAlarm>-Element kann nun alternativ zur Spezifikation eines Zeitpunkts mittels <for> oder <until> einen wiederholten Alarm mittels <repeatEvery> spezifizieren. Das Petrinetz-Muster des Event Handler muss deshalb entsprechend angepasst und erweitert werden.

4.6 Fault Handler

Das Attribut `exitOnStandardFault` regelt das Verhalten im Falle des Auftretens eines (von *bpws:joinFailure* verschiedenen) Fehlers. Für den Fall, dass das Attribut den Wert "yes" besitzt, muss die Abarbeitung des Prozesses bei Auftritt eines Fehlers umgehend abgebrochen werden.

4.7 Compensation Handler

Der wiederholte Aufruf eines Compensation Handler (ob aktiviert oder nicht) darf keinen Fehler mehr produzieren, sondern muss ignoriert werden. Das vorhandene Petrinetz-Muster des Compensation Handler muss dahingehend entsprechend angepasst werden.

5 Zusammenfassung und Ausblick

Ziel dieser Arbeit war es, die Veränderungen der kommenden BPEL-Version WS-BPEL 2.0 im Vergleich zum Vorgänger BPEL4WS 1.1 zu dokumentieren. Dazu wurden die Veränderungen im Detail vorgestellt, wobei der Schwerpunkt der Betrachtung insbesondere auf Änderungen lag, die für eine Anpassung der Petrinetz-Semantik an WS-BPEL 2.0 relevant sein dürften. Des Weiteren wurde eine Art Arbeitsanleitung zur Anpassung der Petrinetz-Semantik formuliert.

Aufbauend auf dieser Arbeit kann nun mit der eigentlichen Anpassung der Petrinetz-Semantik an WS-BPEL 2.0 begonnen werden. Dazu sind die in Kapitel 4 genannten Petrinetz-Muster neu zu entwerfen bzw. zu modifizieren, um eine vollständige formale Semantik für WS-BPEL 2.0 zu erhalten. Es ist jedoch zu prüfen, in wie weit der hier zugrunde gelegte Spezifikationsentwurf von der endgültigen Fassung abweicht.

Liegt eine formale Semantik nach dem Vorbild von Stahl [Sta05] für WS-BPEL 2.0 vor, bietet sich die Untersuchung noch offener Modellierungsfragen, wie z.B. der nach Correlation oder Datenmodellierung, an. Mit einer detailreicheren Abbildung der BPEL-Prozesse in Petrinetze wüchse schließlich auch die Anzahl der verifizierbaren Eigenschaften.

Literaturverzeichnis

- [AAA⁺06] Alexandre Alves, Assaf Arkin, Sid Askary, Ben Bloch, Francisco Curbera, Yaron Goland, Neelakantan Kartha, Canyang Kevin Liu, Vinkesh Mehta, Satish Thatte, Prasad Yendluri, and Alex Yiu. *Web Services Business Process Execution Language Version 2.0*. Committee draft, 16th march 2006, OASIS, March 2006.
- [ACD⁺03] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. *Business Process Execution Language for Web Services, Version 1.1*. Technical report, BEA Systems, International Business Machines Corporation, Microsoft Corporation, May 2003.
- [JC99] Steve DeRose James Clark. *XML Path Language (XPath) Version 1.0*. W3C, W3C Recommendation, 1999. <http://www.w3.org/TR/xpath>.
- [Ley01] Frank Leymann. *WSFL – Web Services Flow Language*. IBM Software Group, Whitepaper, May 2001. <http://ibm.com/webservices/pdf/WSFL.pdf>.
- [Sta05] Christian Stahl. *A petri net semantics for bpel*. Informatik-Berichte 188, Humboldt-Universität zu Berlin, July 2005.
- [Tha01] Satish Thatte. *XLANG – Web Services for Business Process Design*. Microsoft Corporation, Initial Public Draft, May 2001. http://www.gotdotnet.com/team/xml_wsspecs/xlang-c.
- [Whi06] Stephen A. White. *Business Process Modeling Notation (BPMN) Specification, Version 1.0*. Final adopted specification, february 2006, dtc/06-02-01, OMG, February 2006.