Institut für Architektur von Anwendungssystemen (IAAS)

Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart

Studienarbeit Nr. 2173

Bewertung der BPEL-Modellierungsmöglichkeiten im ARIS Toolset

Stefan Varnhorn

Studiengang:	Informatik
Prüfer:	Prof. Dr. Frank Leymann
Betreuer:	Dipl.-Inf. Oliver Kopp
Begonnen am:	4. Februar 2008
Beendet am:	5. August 2008
CR-Klassifikation:	D.2.2, H.4.1, K.1

Inhaltsverzeichnis

Abbildungsverzeichnis

Listingverzeichnis

1 Einleitung

Geschäftsprozesse beschreiben eine zusammenhängende Folge von Aktivitäten, die zur Erreichung eines oder mehrerer Unternehmensziele notwendig sind [Sta06, S. 4]. Das Identifizieren und Optimieren dieser Abläufe kann zu einem deutlichem Geschwindigkeitszuwachs und zur Kostenreduktion führen. Damit tragen effiziente Geschäftsprozesse maßgeblich zum Unternehmenserfolg bei und besitzen eine große strategische Bedeutung für ein Unternehmen.

Nachdem lange Zeit das Ermitteln und Optimieren der Abläufe im Mittelpunkt stand, spielt heutzutage die automatisierbare Ausführung solcher Geschäftsprozesse eine bedeutende Rolle [Kop05, S. 5]. Eine Möglichkeit hierfür ist die Entwicklung umfangreicher Systeme, welche bestimmte Geschäftsprozesse realisieren. Diese Methode hat aber unter anderem den Nachteil, dass die Systeme nicht flexibel genug sind, um Änderung der Prozesse schnell abbilden zu können. Zudem werden viele Aktivitäten innerhalb eines Prozesses durch bereits bestehende Anwendungen realisiert. Eine flexiblere Lösung, welche auch in der Lage ist, die vorhandenen Komponenten wiederzuverwenden, ist der Einsatz von serviceorientierten Architekturen (SOA) an. Diese „brechen die starre Verzahnung von IT-Systemen und Prozessabläufen auf“ [ARISPROD, S. 9]. Dabei werden die Softwaremodule als Dienste verstanden, auf welche mittels einer standardisierten Schnittstelle zugegriffen werden kann. Diese Dienste können nun wiederum einzelnen Aktivitäten zugeordnet werden. Zur automatisierten Ausführung eines Geschäftsprozesses ist neben dieser Zuordnungen von Aktivitäten und Services, noch ein Prozessmodell sowie eine Umgebung, die durch das Modell steuert, notwendig.

Eine technischer Umsetzung dafür bieten nun Web Services. Sie beschreiben eine XML-basierte Schnittstelle für Dienste, mit denen mittels Internetprotokolle kommuniziert werden kann. Um Web Services in einem zusammenhängenden Geschäftsprozess zusammenzufassen, hat sich in der Praxis die *„Web Services Business Process Execution Language“*, kurz BPEL durchgesetzt. Sie ist eine XML-basierte Sprache, welche die Orchestrierung von Web Services ermöglicht. Darunter versteht man die koordinierte und strukturierte Ausführung verschiedener Web Services. Gleichzeitig kann der Prozess wiederum als Web Service nach außen angeboten werden. Die Ausführung eines BPEL-Prozesses ist auf jeder Workflowmaschine möglich, die den BPEL-Standard implementiert.

Aktueller Stand ist die WS-BPEL Version 2.0 [BPEL2.0], welche im April 2007 veröffentlicht wurde. Dessen Vorgänger war die Version 1.1 [BPEL4WS] und trug noch den Namen *„Business Process Execution Language for Web Services”*. Die beiden Versionen sind nicht miteinander kompatibel. Ein ausführlichen Vergleich und die Besprechung der Unterschiede zwischen BPEL 2.0 und BPEL 1.1 befindet sich in [Sch07].

Ein Nachteil von BPEL ist allerdings, dass es eine sehr stark technisch orientierte Sprache ist. Sie benötigt viele Details, die nichts mit der reinen Prozessmodellierung, also dem Abbilden der Geschäftsprozesse, zu tun haben. Zudem bietet die Spezifikation keine grafische Modellierungsmöglichkeit für die Prozesse an. Daher wird BPEL von den fachlichen Verantwortlichen zu der Beschreibung eines Geschäftsprozesses kaum verwendet. Diese nutzen grafische Notationen wie *„ereignisgesteuerte Prozessketten“* [KNS92] oder die *„Business Process Modelling Notation“* [BPMN]. Dies hat zur Folge, dass ein Transformationsschritt benötigt wird, der solche grafischen Modelle nach BPEL überführt (vgl. [Kop05], [Sch08], [Sch07b]).

Jedoch gibt es mittlerweile auch eine Vielzahl an Werkzeugen, mit welchen BPEL-Prozesse grafisch modelliert werden können (vgl. [BKV05]). Da der BPEL-Standard aber eine Visualisierung nicht spezifiziert, sind die Möglichkeiten der Modellierung und Darstellung vollständig von den jeweiligen Herstellern abhängig. Daher unterscheiden sich diese BPEL-Tools teils erheblich in Umfang und Art der Modellierung. Eines dieser Werkzeuge ist das ARIS Toolset.

1.1 Aufgabenstellung

Mit dem ARIS Toolset bzw. ARIS Platform liefert die Firma IDS Scheer [IDS] ein Softwarewerkzeug zur Modellierung und Verwaltung von Geschäftsprozessen. Es unterstützt dabei verschiedene Arten der Modellierung wie ereignisgesteuerte Prozessketten (EPK) oder die Business Process Modelling Notation (BPMN). Zusätzlich wird die Modellierung von BPEL-Prozessen angeboten, welche Gegenstand dieser Arbeit ist. Dabei soll die Fähigkeit der ARIS-Software, BPEL-Prozesse zu modellieren und verwalten, dargestellt und bewertet werden.

2 Hintergrund

In diesem Kapitel wird zunächst der Begriff ARIS erläutert und die darauf beruhende Softwarepakete der Firma IDS Scheer vorgestellt. Dazu wird noch angegebenen auf welcher Software und zugehörigen Standards diese Studienarbeit aufbaut.

2.1 ARIS

Die *„Architektur integrierter Informationssysteme“* (ARIS) [Sch97] beschreibt ein Konzept zur Abbildung von Prozessen in betriebliche Informationssysteme. Sie gibt im Wesentlichen vier verschiedene Sichten an, aus denen ein Gesamtsystem aufgebaut werden kann: *Organisationssicht*, *Funktionssicht*, *Datensicht* und *Steuerungssicht*. Die Organisationssicht gibt die Aufbauorganisation des Unternehmens dar. Die Funktionssicht enthält die Aktivitäten, die innerhalb eines Prozesses durchgeführt werden. Diese können Daten verwenden, die in Datensicht festgelegt sind. Die Steuerungssicht integriert die anderen Sichten innerhalb eines Prozessablaufes.

Die einzelnen Sichten sind jeweils noch aus *Fachkonzept*, *DV-Konzept* und *Implementierung* aufgebaut. Das Fachkonzept beschreibt die in der jeweiligen Sicht verwendeten Objekte in einer weniger technischen Art (Beispiel für Datensicht: *Entity-Relationship-Diagramme*). Sie werden im DV-Objekt näher zu der verwendeten Technik festgelegt (Beispiel: *Tabellen*). Die Implementierungsebene beschreibt die technische Umsetzung (Beispiel: *Datenbanksystem*).

2.2 ARIS Platform

Die Firma IDS Scheer [IDS] liefert Softwarewerkzeuge zur Erstellung und Verwaltung von Geschäftsprozessen, die auf dem ARIS-Konzept aufbauen. Die Werkzeuge werden innerhalb der *„ARIS Platform“* zusammengefasst. Diese beinhaltet eine Menge an integrierten Softwareprodukten, die das gesamte Management der Geschäftsprozess eines Unternehmens unterstützen. Sie umfasst dabei von der Festlegung der Strategie über den Entwurf der Prozesse und deren Umsetzung in IT-Modelle, sowie der Überwachung ausgeführter Prozesse, alle Schritte des Geschäftsprozessmanagements [ARISPROD].

Diesen Phasen entsprechend ist die „ARIS Platform“ in vier Module gegliedert, welche die dazugehörenden Softwareprodukte beinhaltet. Diese werden in nachfolgend mit einer kurzen Beschreibung ihrer Hauptfunktionen vorgestellt [ARISPROD].

- „ARIS Strategy Platform“

  Sie dient der Definition von Geschäftsstrategien und daraus ableitbaren Prozessen,

sowie dem Überprüfen der Zielerreichung.

- „ARIS Design Platform“

  Software zur Modellierung von Geschäftsprozessen, deren Simulation und Optimierung, sowie dem Verwalten bestehender IT-Architekturen.

- „ARIS Implementation Platform“

  Enthält Werkzeuge zur Überführung von Geschäftsprozessen in ausführbare Modelle und zum Aufbau serviceorientierte Architekturen.

- „ARIS Controlling Platform“

  Dient unter anderem zur Überwachung von laufenden Geschäftsprozessen.

Die aus [ARISWP] übernommene Abbildung 1 zeigt alle zur „ARIS Platform“ gehörenden Softwarewerkzeuge und deren Zuordnung zu den vier Modulen.

Abbildung 1: Software der "ARIS Platform" (aus ARIS System White Paper [ARISWP]

2.3 Hinweise zur Studienarbeit

Für die Erstellung dieser Arbeit wird der *„ARIS SOA Designer“* in der *Version 7.02* verwendet. Hiermit wurden alle abgebildeten Modelle erstellt. Dieser unterstützt laut Hersteller den BPEL Standard in der Version 1.1 [BPEL4WS]. Wenn im Folgenden von BPEL oder der „Business Process Execution Language“ die Rede ist, so wird dabei, sofern nicht explizit anders erwähnt, die Version 1.1 gemeint.

3 BPEL-Modellierung mit der ARIS Platform

In diesem Kapitel wird die BPEL-Modellierung mit dem „ARIS SOA Designer“ vorgestellt und der aus den Modellen generierte BPEL-Code gegen die Spezifikation geprüft. Dazu werden zunächst die grundlegenden Eigenschaften der Modellierung mit dem ARIS-Werkzeug, sowie die Importfunktionen für WSDL- und XML Schema-Dokumente beschrieben.

Die BPEL-Modellierungsmöglichkeit des „ARIS SOA Designer“ ist eigentlich nicht dazu gedacht, BPEL-Prozesse von Grund auf neu zu definieren. Vielmehr soll das Werkzeug für die Erstellung einer serviceorientierten Architektur nach der Vorgehensweise, wie sie in [Klü07] beschrieben ist, dienen. Diese sieht vor, dass der Prozess zunächst auf fachlicher Ebene modelliert wird. Als Notation hierzu bieten sich ereignisgesteuerten Prozessketten an. Die darin vorkommenden Aktivitäten können an Services gebunden werden. Dazu verwendet der „ARIS SOA Designer“ ein Repository, in welchem Web Services gespeichert sind. Diese, um konkrete Dienste erweiterte, Prozessmodelle werden dann durch das Tool in BPEL-Modelle überführt. Erst hier kommt nun die BPEL-Modellierungsfähigkeit des „ARIS SOA Designers“ ins Spiel. Damit können den Prozessen technische BPEL-Eigenschaften, wie Fehler- oder Ereignisbehandlung zugefügt werden. Aus diesen BPEL-Prozessmodellen wird dann der BPEL-Code erzeugt, welcher in einer entsprechenden Umgebung ausgeführt werden kann. Trotz der eigentlich anders gedachten Vorgehensweise, wird im Rahmen dieser Studienarbeit dennoch die Möglichkeit einen BPEL-Prozess von Grund auf zu modellieren, dargestellt und untersucht.

3.1 Grundlagen der BPEL-Modellierung

3.1.1 Modelltypen

Für die Modellierung von BPEL-Prozessen werden im ARIS Werkzeug zwei Modelltypen verwendet: das *„BPEL Process Model“* und das *„BPEL Allocation Diagram“*. Der Modelltyp „BPEL Process Model“ kann zur Darstellung eines gesamten Geschäftsprozesses eingesetzt werden. Um bestimmte Elemente des BPEL-Prozesses genauer zu modellieren, wird der Modelltyp „BPEL Allocation Diagram“ verwendet. Dies sind Zuordnungsdiagramme, die einem BPEL-Element zugewiesen sind und dieses mit Details versehen können. Solche Verfeinerungen einzelner Elemente können auch direkt im Prozessmodell erfolgen. Allerdings dienen die zusätzliche Diagramme der Reduzierung der Komplexität des gesamten Prozessmodells. Die Auslagerungsmöglichkeit von Details in andere Diagramme sorgt dafür dass der eigentlichen Prozess noch besser lesbar bleibt und nicht mit zu vielen technischen BPEL-Informationen überladen ist.

Zur Veranschaulichung der Vorteile, welche durch den Einsatz dieser zwei Modellarten

entstehen, soll die Abbildung 2 beitragen. Sie zeigt auf der linken Seite ein Prozessmodell, welches eine einfache Abfolge verschiedener Aktivitäten beinhaltet. Dem *„ProcessStart"*-Element `'Reisebuchung'` des Prozesses wurde das Allokationsdiagramm im rechten Teil der Abbildung zugeordnet. Dieses enthält BPEL-spezifische Details des Prozesses, wie die Definition eines Partner Links `'Airline'` und einer Variable `'Kundenname'`. Dass einem Objekt eine weiteres Diagramm zugewiesen ist, zeigt das kleine Symbol rechts unterhalb des Symboles.

Anhand dieses recht kleinen Beispiels wird schon deutlich, wie die Komplexität eines Prozessmodells durch den Einsatz der zwei Modelltypen reduziert wird. Der reine Ablauf des Prozesses ist immer noch gut lesbar. Die BPEL-Details die zu dem Prozessmodell gehören, befinden nicht innerhalb dieses Modells, sondern in einem zusätzlichen Diagramm. Im Beispiel aus Abbildung 2 wurde nur dem Objekt `'Reisebuchung'` eine Allokationsdiagramm zugewiesen. Ebenso können sämtliche andere Objekte durch weitere zugeordnete Diagramme verfeinert werden. Dadurch bestehen mit dem „ARIS SOA Designer" modellierte BPEL-Prozesse meist aus einer Vielzahl kleinerer Diagramme.

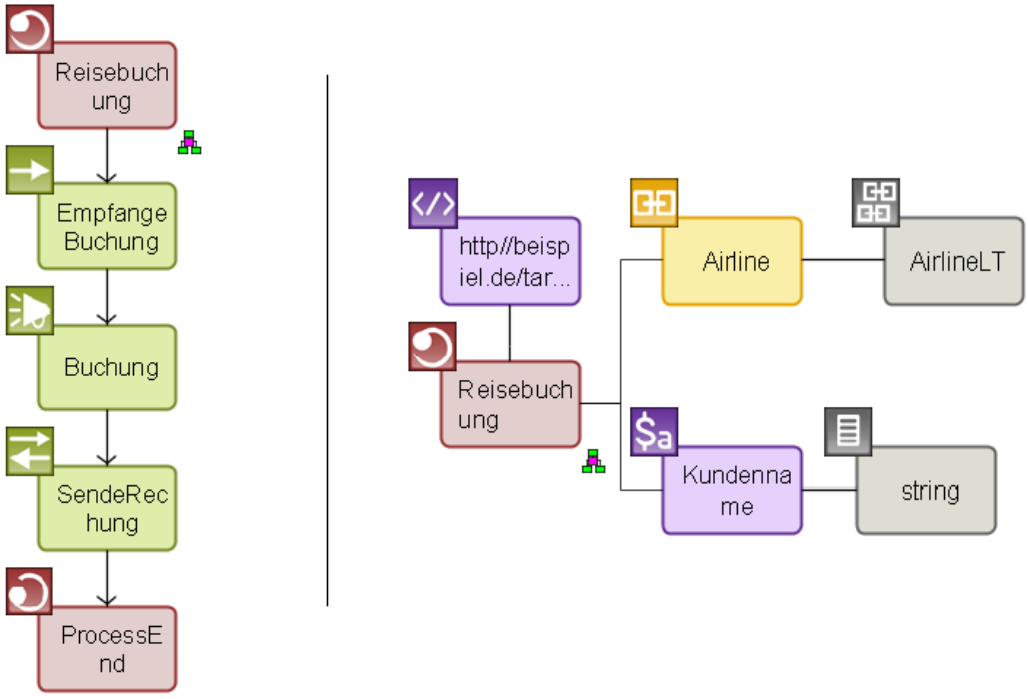

Abbildung 2: Beispiel für die beiden Modelltypen des ARIS SOA Designer

3.1.2 BPEL-Elemente

Um einen BPEL-Prozess zu modellieren, existieren im ARIS Tool verschiedene Elementtypen, die innerhalb eines Modells einsetzbar sind. Diese repräsentieren jeweils bestimmte Konstrukte der Sprache BPEL und werden durch verschiedene Symbole dargestellt. Ein Teil der Symbolauswahl ist in Abbildung 3 wiedergegeben. Zur Verwendung eines solchen

Elementes, wird das entsprechende Symbol in den Modellierungsbereichs des „ARIS SOA Designers“ eingefügt. Dies erzeugt automatisch ein entsprechendes BPEL-Objekt in der Datenbank des ARIS Werkzeugs. Dieses Objekt kann mehrmals in einem Prozessmodell bzw. den zugehörigen Zuordnungsdiagrammen benutzt werden. Außerdem können Objekte eines Prozessmodells in anderen Modellen verwendet werden. Bei den meisten Elementypen können noch bestimmte Attribute entsprechend der BPEL-Spezifikation über ein Menü gesetzt werden. Bei welchen Elementen diese Attribute von Bedeutung sind, wird im Laufe dieser Arbeit im entsprechenden Kontext erläutert.

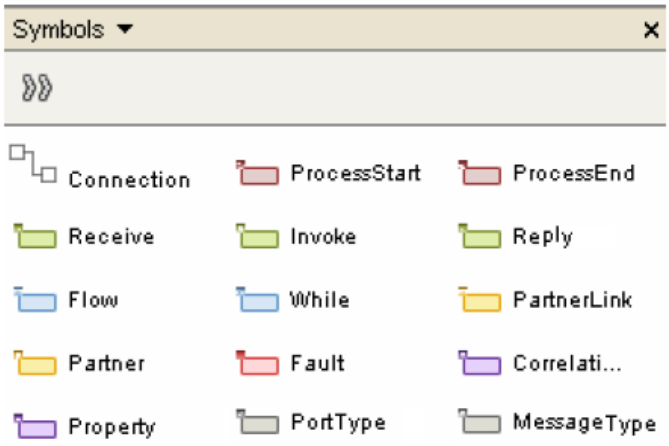

Abbildung 3: Ausschnitt aus der BPEL-Symbolpalette des ARIS Tools

Um einen Prozess zu modellieren, müssen die verwendeten Objekte zueinander in Beziehung gesetzt werden. So bedeutet im Beispiel aus Abbildung 2 der Pfeil zwischen den Symbolen `'EmpfangeBuchung'` und `'Buchung'`, dass die Aktivität `'EmpfangeBuchung'` vor der Aktivität `'Buchung'` durchgeführt wird. Dies entspricht der Modellierung *einer Sequence*. Eine Beziehung zwischen zwei Objekten wird in der Regel durch eine Linie oder einen Pfeil dargestellt. Eine solche Verbindung besitzt immer auch einen bestimmten Typ. Für diesen Typ, also der Beziehungsart zweier Objekte, können teilweise mehrere Möglichkeiten existieren. Meist geben die Elementtypen der verbundenen Objekte direkt den Beziehungstyp vor. Ist dies nicht der Fall, hat der Benutzer die Möglichkeit den Typ entsprechend der zu modellierenden Eigenschaft selbst auszuwählen. Details dazu werden in den nachfolgenden Abschnitten bei der Beschreibung der jeweiligen Elementtypen erläutert.

3.1.3 Verwaltung der Modelle

Zur Verwaltung und Ordnung der Prozessmodelle bietet der „ARIS SOA Designer“ eine einfache Dateistruktur an. Es können Ordner (Gruppen) angelegt werden, welche die darin definierten Modelle und Objekte beinhalten. Die Gruppen können in ihrer Hierarchie beliebig tief sein. Auf diese Weise können die Prozessmodelle beispielsweise nach ihrem logischen Verwendungszweck strukturiert werden.

3.1.4 BPEL-Export

Damit die modellierten zur Ausführung gebracht werden soll,, müssen die modellierten Prozesse in das BPEL-Format exportiert werden. Dazu bietet der „ARIS SOA Designer“ eine Exportfunktion an, die aus einem Prozessmodell und allen zugeordneten Allokationsdiagrammen entsprechenden BPEL-Code generiert. Diese erzeugt dabei eine komprimierte zip-Datei, welche eine BPEL-Datei des Prozesses und eine WSDL-Datei, die Schnittstelle des Prozesses als Web Service beschreibt, enthält. Außerdem können innerhalb eines Prozessmodells auch noch neue Web Services, die vom Prozess über Partner Links verwendet werden, modelliert sein. In diesem Fall wird auch noch einen Ordner 'imports' erzeugt, der für jeden neu modellierten Web Service eine generierte WSDL-Beschreibung enthält. Das exportierte Zip-Archiv hat also folgenden grundsätzlichen Aufbau:

- Prozessmodell.bpel
- Prozessmodell.wsdl
- imports\PartnerService.wsdl ?

3.1.5 Importfunktionen für Services und XML-Schemata

Ein BPEL-Prozess kommuniziert im Allgemeinen mit anderen Web Services. Damit solche Dienste nicht jedes Mal innerhalb der Prozessmodellierung neu modelliert werden müssen, gibt es auch die Möglichkeit WSDL-Beschreibungen von Services zu importieren. Diese Importfunktion erzeugt innerhalb der Gruppe in die sie importiert wurde eine neue Untergruppe. Diese trägt den Namen des in der WSDL-Datei angegebenen Zielnamensraumes und enthält Objekte, welche für die BPEL-Modellierung benötigt werden. Dies sind beispielsweise Objekte für Porttypen oder Nachrichtentypen, die in der WSDL-Beschreibung definiert sind. Die so erzeugten Objekte repräsentieren also die entsprechenden Definitionen der importierten WSDL-Datei. Sie können für die Modellierung eines Prozesses genutzt werden. Ihre Verwendung ist dabei genau so, wie die der selbst erstellten Objekte. Sie referenzieren intern aber den Zielnamensraum der WSDL-Beschreibung, welcher beim späteren Export nach BPEL entsprechend umgesetzt wird.

Zudem werden so importierte WSDL-Dateien, sofern sie eine `<service>` Definition enthalten, auch in das Service-Repository des „ARIS SOA Designer“ eingefügt. Daraus können die Dienste bei einer EPK-Modellierung bestimmten Aktivitäten zugewiesen werden (vgl. [Klü07]).

Nicht nur Dienste, sondern auch XML Schema Datentypen können importiert und in einem BPEL-Prozessmodell verwendet werden. Dazu wird eine XSD-Datei benötigt, die die Definition der Datentypen enthält. Der XSD-Import erzeugt, wie der WSDL-Import, eine Gruppe die den Namen des entsprechenden Namensraumes trägt und Objekte, welche

diese Datentypen repräsentieren. Sie können analog zu WSDL-Objekten für die Prozesserstellung genutzt werden.

3.2 Grundstruktur eines BPEL-Prozesses

Die Definition eines BPEL-Prozess besteht grundsätzlich aus fünf Abschnitten: den *Partner Links* und *Partnern*, den *Variablen*, den *Correlation Sets*, den *Handlers* für die Ereignis-, Fehler- und Kompensationsbehandlung, sowie der Beschreibung des Prozessablaufs durch Aktivitäten. Auf die Bedeutung und Aufgaben der einzelnen Bestandteile wird im Rahmen der vorliegenden Studienarbeit nicht detailliert eingegangen. Eine kurze, teils unvollständige Erklärung der einzelnen BPEL-Konstrukte befindet sich jeweils bei der Beschreibung der zugehörigen Modellierungsmöglichkeit. Ausführliche Erläuterungen finden sich in der Spezifikation [BPEL4WS] der „Business Process Execution Language“. Nachfolgend ist in Listing 1 eine vereinfachte Darstellung der Grundstruktur eines BPEL-Prozesses gegeben, welche aus den oben genannten Bestandteilen aufgebaut ist.

```
<proces xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">

   <partnerLinks>?
      <partnerLink/>+
   </partnerLinks>
   <partners>?
      <partner/>+
   </partners>

   <variables>?
      <variable>+
   </variables>

   <correlationSets>?
      <correlationSet/>+
   </correlationSets>

   <faultHandlers>?
      <catch faultName="qname"? faultVariable="ncname"?>*
         activity
      </catch>
      <catchAll>?
         activity
      </catchAll>
   </faultHandlers>

   <compensationHandler>?
      activity
   </compensationHandler>

   <eventHandlers>?
      <onMessage>
         activity
      </onMessage>
      <onAlarm>*
```

```
            activity
        </onalarm>
    </eventHandlers>

    activity

</process>
```

Listing 1: Grundstruktur eines BPEL-Prozesses

Der „ARIS SOA Designer“ bietet für diese Bestandteile jeweils eigene Elementtypen, die zur Modellierung verwendet werden. Ein BPEL-Prozess, der die Grundbestandteile beinhaltet, ist in der Abbildung 4 dargestellt. Diese Darstellung dient nur der Veranschaulichung der Grundstruktur eines BPEL-Prozessmodells und dem Aufzeigen der verschiedenen Elementtypen sowie deren Verbindungen. Die genaue Bedeutung der einzelnen Elemente und ihre Verwendung in einem BPEL-Prozess wird in den nachfolgenden Abschnitten ausführlich erläutert.

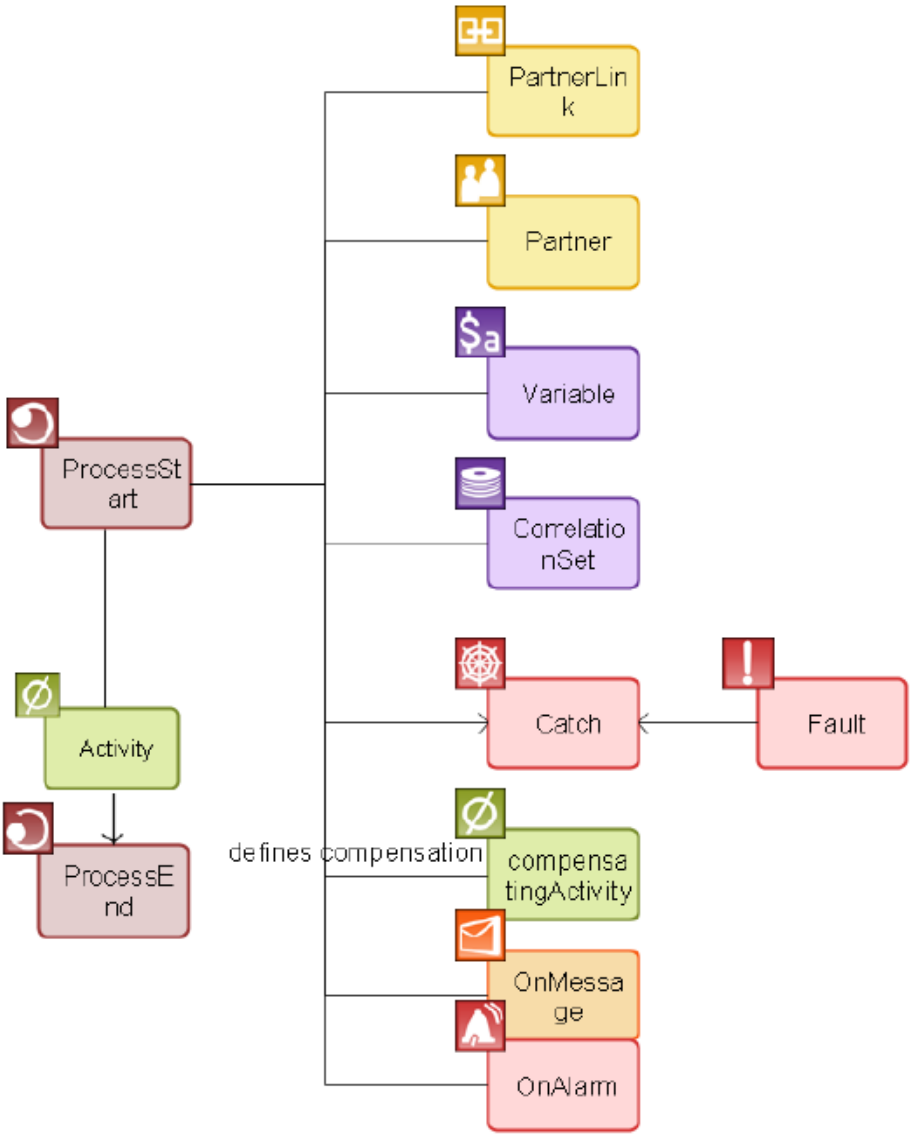

Abbildung 4: Aufbau eines BPEL-Prozessmodelles

Die Beschreibung des Prozessablaufs befindet sich zwischen dem *„ProcessStart“*- und dem *„ProcessEnd“*-Element. Sie besteht in diesem Fall nur aus einer Aktivität. Die Definition der weiteren in Listing 1 vorkommenden Konstrukte erfolgt durch die Verbindungen zwischen den jeweiligen Elementtypen und dem *„ProcessStart“*-Element. Wie in Abschnitt 3.1.3 genannt, besitzt jede Beziehung einen bestimmten Typ. Die Verbindung zwischen dem *„ProcessStart“*-Element und den ersten vier Elementen, also den *„PartnerLink“*-, *„Partner“*-, *„Variable“*- und *„CorrelationSet“*-Elementen ist vom Typ *„defines“*. Für Beziehungen zwischen diesen Element und einem *„ProcessStart“*-Element ist auch nur dieser Typ möglich. Im Gegensatz dazu gibt es für die Verbindung zwischen einem *„ProcessStart“*- und einem *„Aktivität“*-Element mehrere Möglichkeiten. Im abgebildeten Modell ist beispielsweise das Objekt `'Activity'` über *„occurs after“* (bzw. je nach Leserichtung *„occurs before“*) mit dem Objekt `'ProcessStart'` verbunden. Dagegen ist die Beziehung zwischen `'ProcessStart'` und `'compensatingActivity'` vom Typ *„defines Compensation“*. Eine Art beschreibt dabei den Prozessablauf, während die andere einen *Compensation Handler* definiert. Die weiteren in Abbildung 4 verwendeten Elementtypen *„Catch“*, *„onMessage“* und *„onAlarm“* definieren die *Fault* bzw. *Event Handler* des Prozesses.

3.2.1 Wurzelelement eines BPEL-Prozesses

Das *„ProcessStart“*–Element stellt den Startknoten eines BPEL-Prozessmodells dar. Mit ihm werden die top-level Attribute eines Prozesses festgelegt, wie sie der BPEL-Standard definiert [BPEL4WS, S26]. Es ist in Listing 2 dargestellt.

```
<process name="ncname"
         targetNamespace="uri"
         queryLanguage="anyURI"?
         expressionLanguage="anyURI"?
         suppressJoinFailure="yes|no"?
         enableInstanceCompensation="yes|no"?
         abstractProcess="yes|no"?
         xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
```

Listing 2: BPEL-Syntax des Prozess-Elementes mit den top-level-Attributen

Auf die Bedeutung der einzelnen Attribute wird hierbei nicht detailliert eingegangen. Diese kann unter [BPEL4WS, S26] nachgeschlagen werden. Für die optionalen Attribute gibt die Spezifikation folgende Default-Werten vor.

- expressionLanguage = `"http://www.w3.org/TR/1999/RECxpath-19991116"`
- queryLanguage = `"http://www.w3.org/TR/1999/REC-path-19991116"`
- suppressJoinFailure = `"no"`
- enableInstanceCompensation = `"no"`
- abstractProcess = `"no"`

Die Verwendung des *„ProcessStart“*-Elementes im „ARIS SOA Designer“ wurde in der vorherigen Abbildungen schon kurz aufgezeigt. Es dient der Definition des BPEL-Prozess-

modells und zeigt dessen Anfang. In Abbildung 5 wird ein „leerer“ Prozess modelliert, welcher ausschließlich das *„ProcessStart“* Element mit dem Namen `'Name_Of_Process'` besitzt und ein damit verbundenes *„Namespace“*-Element mit dem Namen `'URI_of_TargetNamespace'`. Damit wird nur der BPEL-Prozess und eine zugehöriger Zielnamensraum festgelegt.

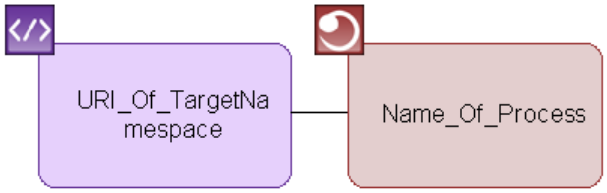

Abbildung 5: Prozessstartelement

Die BPEL-Export-Funktion generiert daraus den folgenden BPEL-Code aus Listing 3.

```
<process xmlns="http://schefmas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:tns="URI_Of_TargetNamespace"
         expressionLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116"
         name="Name_Of_Process"
         queryLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116"
         targetNamespace="URI_Of_TargetNamespace"/>
```

Listing 3: Aus Beispielmodellierung generierte <prozess>-Element

Dabei werden die optionalen Attribute also korrekt mit den spezifizierten Standardwerten besetzt. Sie können über die gleichnamigen BPEL-Attribute des *„ProcessStart“*-Elementes selbst den eigenen Anforderungen entsprechend gesetzt werden.

Bei der Verwendung eines *„ProcessStart“*-Elementes ohne ein dazu in Beziehung stehendes *„Namespace“*-Element gibt die BPEL-Export-Funktion eine Warnung aus, dass kein Namensraume für das Prozessziel gefunden wurde und deshalb der Standardnamensraum `"http://www.ids-scheer.com/bpel"` verwendet wird.

Bei der Definition eines BPEL-Prozesses mit den entsprechenden Attributen erfüllt der „ARIS SOA Designer“ also, wie hier aufgezeigt, den BPEL-Standard.

3.3 Partner Links und Partner

3.3.1 Partner Links

Mit *Partner Links* wird in BPEL festgelegt, wie ein Prozess mit seinen Partnern verbunden werden kann. Jeder Partner Link ist über einen *Partner Link Type* beschrieben, welcher Porttypen und Rollen angibt, über die die Kommunikation zu anderen Services realisiert wird. Eine ausführliche Erklärung findet sich in der BPEL-Spezifikation [BPEL4WS, S32ff]. Die PartnerLinkTypes werden in einer WSDL-Datei definiert, welche entweder die verwende-

ten Servicebeschreibung enthält oder darauf verweist. Der Festlegung Partner Links selbst erfolgt innerhalb des BPEL-Prozesses. Die Syntax für die beiden Definitionen ist in Listing 4 dargestellt.

```
<definitions name="ncname" targetNamespace="uri"
       xmlns="http://schemas.xmlsoap.org/wsdl/"
       xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">

   <plnk:partnerLinkType name="ncname">
      <plnk:role name="ncname">
         <plnk:portType name="qname"/>
      </plnk:role>
      <plnk:role name="ncname">?
         <plnk:portType name="qname"/>
      </plnk:role>
   </plnk:partnerLinkType>
</definitions>



<partnerLinks>
   <partnerLink name="ncname" partnerLinkType="qname"
                myRole="ncname"? partnerRole="ncname"?>+
   </partnerLink>
</partnerLinks>
```

Listing 4: BPEL-Syntax der Partner Link und PartnerLinkType Definition

Die grundsätzliche Modellierung der Definition eines PartnerLinks innerhalb eines Prozesses ist in Abbildung 6 dargestellt. Dazu wird ein *„PartnerLink“*-Element verwendet, welches mit dem *„ProcessStart“*-Element des entsprechenden Prozesses verbunden ist. Zusätzlich wird es mit einem *„PartnerLinkType“*- und einem *„PortType“*-Element verbunden. Über die Attribute der Beziehung zu dem *„PortType“*-Element kann das `myRole` bzw. `partnerRole`-Attribut des PartnerLinks festgelegt werden. Diese Beziehung ist vom Typ *„links PortType“* und besitzt dazu die Attribute *„Role type“* und *„Connection role“* (siehe Abb. 6). Dabei wird über *„Connection role“* ausgewählt, ob der Prozess selbst oder der Partner eine bestimmte Rolle übernimmt. Der Name dieser Rolle wird durch das Attribut *„Role type“* festgelegt.

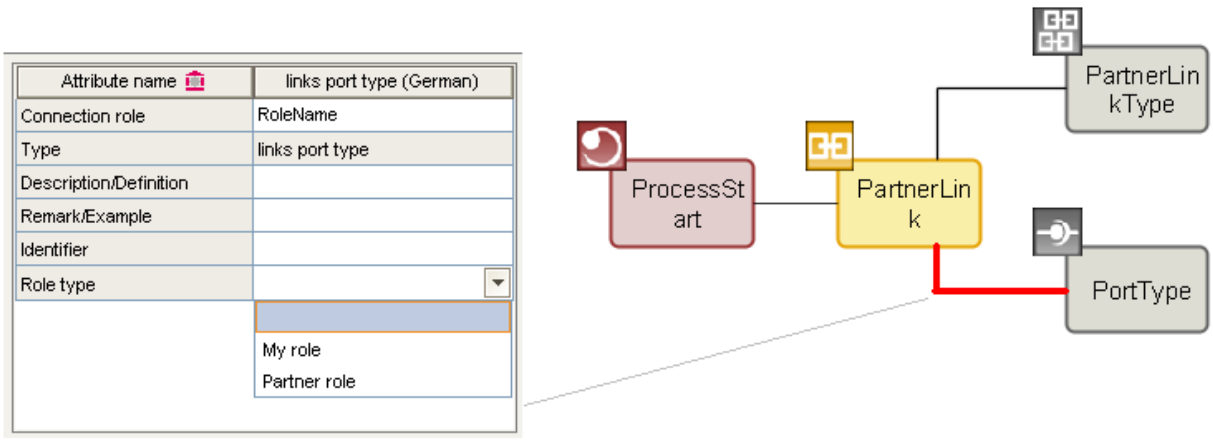

Abbildung 6: Modellierung einer Partner Link Definition

Um den Prozess selbst als Web Service mit einem bestimmten PartnerLinkType nach außen anzubieten, muss die WSDL-Beschreibung des Prozesses jene PartnerLinkType-Definition beinhalten. Dies wird modelliert, indem das Attribut „*Define Process Partner Link*“ des *„PartnerLink“*-Elementes (nicht in der Abbildung dargestellt) auf 'wahr' gesetzt wird.

Damit die Definition eines Partner Links innerhalb eines Prozesses auch sinnvoll ist und weiter verwendet werden kann, müssen die verwendeten Web Services bekannt sein. Dazu kann eine importierte WSDL-Beschreibung und die daraus erzeugten Objekte zur Modellierung der Partner Links verwendet werden. Eine weitere Möglichkeit ist es, die Operationen und Porttypen des Services direkt mit dem „ARIS SOA Designer“ zu modellieren. Nachfolgend werden beide Möglichkeiten anhand eines Beispieles dargestellt.

In Abbildung 7 befindet sich die Beispielmodellierung einer Partner Link Definition, bei der ein zugehöriger Services mit modelliert wird. Im oberen Abschnitt ist dabei ein Prozessmodell, mit der zuvor erläuterten Definition eines Partner Links abgebildet. Die Rolle wurde dabei entsprechend der zuvor beschriebenen Vorgehensweise auf `partnerRole="loanService"` gesetzt. Der untere Teil zeigt ein Zuordnungsdiagramm, welches den Porttyp genauer beschreibt. Das *„PortType“*- Element `'loanServicePT'` ist dazu mit einer Operation `'request'` verbunden. Dieses *„Operation“*- Element ist seinerseits wieder mit den *„Parameter“*-Elementen `'in'` und `'out'`, sowie einem *„Fault“*-Element in Beziehung gesetzt wurden. Somit wird modelliert, dass der `'loanServicePT'` die Operation `'request'` enthält. Diese besitzt als Parameter Nachrichten, welche vom den angegebenen Nachrichtentypen `'creditInformationMessage'` bzw. `'approvalMesage'` sind. Außerdem kann die Operation noch einen Fehler `'unbleToHandlerequest'` des Nachrichtentyps `'errorMessage'` verursachen. Bei den *„Parameter“*-Elementen muss die Richtung, also ob es sich um eine Input- oder eine Output-Message der Operation handelt, beachtet werden. Dazu besitzt das *„Parameter“*-Element das Attribut *„Direction“*, welches zur Auswahl der Richtung dient. Die Definition der verwendeten Nachrichtentypen, also der Aufbau aus bestimmten Teilen (MessageParts) und zugehörigen Datentypen, werden in eigenen Zuordnungsdiagrammen noch genauer spezifiziert. Diese sind in der Abbildung nicht angegebenen. Die Definitionen der Nachrichtentypen wird im folgenden Abschnitt im Zusammenhang der Definition von Variablen näher vorgestellt.

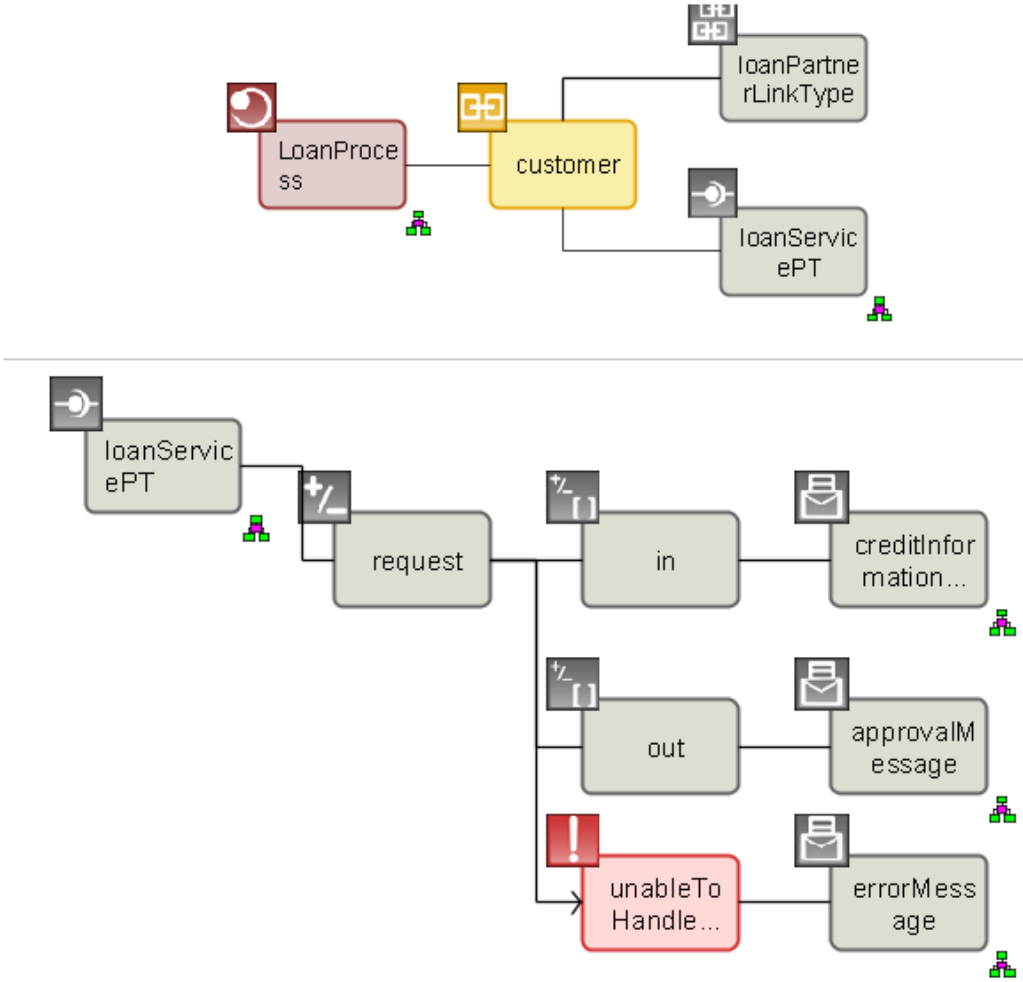

Abbildung 7: Beispielmodellierung eines Partner Links und dazugehörigem Service

Die Exportfunktion generiert aus dieser Beispielmodellierung die in Listing 5 angegebene WSDL-Datei. Dabei wurde auch die Warnung „`Warning: RoleName loanService is not defined by a PartnerLinkType of PartnerLink customer.`“ ausgegeben.

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
      xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      name="loanPartnerLinkType"
      targetNamespace="http://www.ids-scheer.com/bpel">


  <portType name="loanServicePT">
    <operation name="request">
     <input message="tns:creditInformationMessage" name="request_Request"/>
     <output message="tns:riskAssessmentMessage" name="request_Response"/>
     <fault message="tns:errorMessage" name="unableToHandleRequest"/>
    </operation>
  </portType>
```

```
  <plnk:partnerLinkType name="loanPartnerLinkType"/>
</definitions>
```

Listing 5: Aus Beispielmodell generierte Beschreibung eines Web Service

Diese erzeugte WSDL-Beschreibung eines Services definiert den modellierten Porttyp mit dessen Operation und Nachrichten korrekt. Allerdings wird der PartnerLinkType `"loanPartnerLinkType"` ohne die zugehörige `<plnk:role>` und `<plnk:portType>` Elemente generiert. Dies entspricht auch der Fehlermeldung, wonach der Rollennamen nicht durch den PartnerLinkType definiert ist. Die Modellierung erfolgte nach der Vorgabe aus dem Benutzerhandbuch [ARISDS70, S. 37]. Allerdings ließ sich nicht herausfinden, wie die fehlende Definition einer PartnerLinkTypes richtig erfolgt. Es besteht keine Möglichkeit ein *„PartnerLinkType“*- und ein *„PortType“*-Element in Beziehung zu setzen, welches der intuitive Ansatz aufgrund der Spezifikation (Listing 4) wäre. Außerdem gibt es keine Attribute, welche eine Definition des PartnerLinkTypes ermöglichen.

Die beim Export ebenfalls generierte BPEL-Datei des Prozesses beinhaltet dagegen die Definition eines Partner Links, welcher die BPEL-Syntax erfüllt. Sie ist in Listing 6 angegeben. Hierbei wird der Partner Link mit den, der Modellierung entsprechend, korrekten Namen, PartnerLinkType und Rollen definiert.

```
<partnerLinks>
   <partnerLink name="customer" partnerLinkType="tns:loanPartnerLinkType"
                partnerRole="loanService"/>
</partnerLinks>
```

Listing 6: Aus Beispielmodell generierte Partner Link Definition (eigenes Modell)

Nachdem im vorangegangen Beispiel der Web Service neu modelliert wurde, wird nun in Abbildung 8 ein Partner Link Definition modelliert, bei welcher der Web Service importiert ist. Dazu wird zunächst die Beschreibung eines einfachen Web Services importiert, wobei unter anderen das im Modell verwendete *„PartnerLinkType“*-Objekt `'printLinkType'` und das *„PortType“*-Objekt `'Print'` erzeugt wird. Mit ihnen wurde die Definition des Partner Link `'printServicePartnerLink'` modelliert. Die Attribut *„Role type“* und *„Connection role“* des *„PartnerLink“*-Elementes wurde dabei auf `partnerRole` bzw `printService` gesetzt.

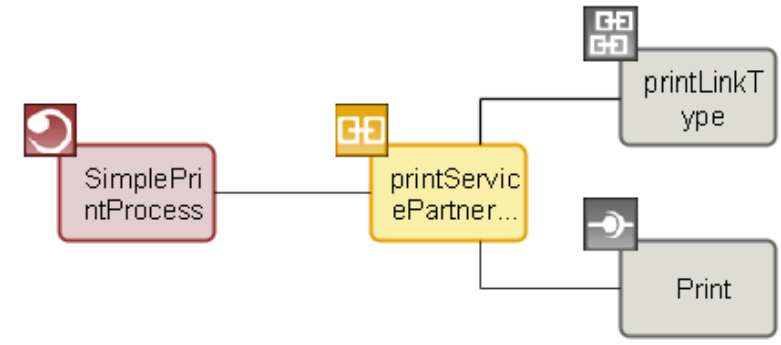

Abbildung 8: Beispielmodellierung einer Partner Link Definition

Der für die Überprüfung der Partner Link Definition relevante Ausschnitt aus dem exportierten BPEL-Prozess ist nachfolgend in Listing 7 angegeben

```
<partnerLinks>
   <partnerLink name="printServicePartnerLink" partnerRole="printService"/>
</partnerLinks>
```

Listing 7:Aus Beispielmodell generierte Partner Link Definition (importiert)

Der Name und die Rolle sind dabei korrekt erzeugt wurden. Jedoch wird der PartnerLinkType `'printLinkType'` nicht übernommen. Damit liefert die Export-Funktion auch in diesem Fall keinen korrekten BPEL-Code.

3.3.2 Geschäftspartner

Die Kommunikation mit einem Geschäftspartner besteht häufig aus mehr als nur einem Kommunikationskanal. Um alle Anforderungen an einen Geschäftspartner anzugeben, gibt es in BPEL das Element `<partner>`. Dieses Element beinhaltet eine Menge an Partner Links, die der Partner anbieten muss [BPEL4WS, S. 35]. Die Syntax ist in Listing 8 angegeben.

```
<partners>
   <partner name="ncname">+
      <partnerLink name="ncname"/>+
   </partner>
</partners>
```

Listing 8: BPEL-Syntax der Partner Definition

Die Modellierung mit dem ARIS SOA Designer erfolgt analog zur dieser Definition. Ein *„Partner“*-Element wird mit sämtlichen *„PartnerLink“*-Objekten, die der Partner erfüllen soll, in Beziehung gesetzt. Um die Definition dieses Partners einem Prozess zuzuordnen, wird das *„Partner“*-Element noch mit dem entsprechenden *„ProcessStart“*- Element verbunden.

Die Abbildung 9 zeigt zum Beispiel die Definition eines Geschäftspartners `'Seller-Shipper'`, der die Partner Links `'Seller'` und `'Shipper'` verwenden soll.

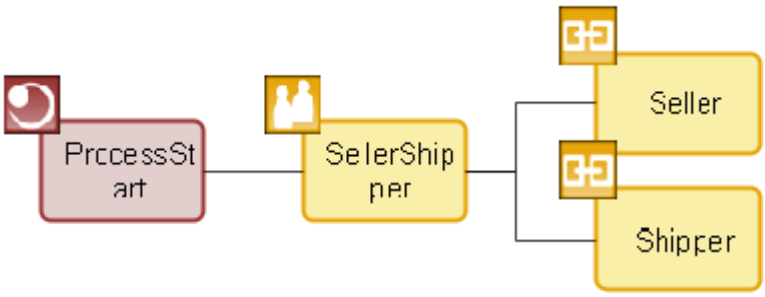

Abbildung 9: Beispielmodellierung einer Partner Definition

In Listing 9 ist die aus dem obigen Modell exportierte Partner Definition angegeben. Sie ist

korrekt und erfüllt die Spezifikation.

```
<partners>
   <partner name="SellerShipper">
      <partnerLink name="Seller"/>
      <partnerLink name="Shipper"/>
   </partner>
</partners>
```

Listing 9: Aus Beispielmodell generierte Partner Definition

Würde man bei dieser Modellierung die *„PartnerLink“*-Elemente weglassen, also das *„Partner“*-Element nur mit den *„ProcessStart“* verbinden, wird daraus das Element `<partner name="SellerShipper"></partner>` erzeugt. Dies entspricht zwar dem Modell, aber nicht der BPEL-Spezifikation (vgl. Listing 8). Eine entsprechende Warnung wird beim Export nicht angegeben, obwohl dies eventuell recht sinnvoll wäre.

Dass bei korrekter Modellierung auch korrekter BPEL-Code erzeugt wird, bei Weglassen einer Elementverbindung (also falscher Modellierung) aber ohne Warnung falscher BPEL-Code erzeugt wird, tritt häufig auf. Darauf wird im Folgenden meist nicht weiter eingegangen, sondern die jeweils richtige Modellierung beschrieben.

3.4 Variablen

In BPEL werden Variablen verwendet um Zustände zu speichern. Sie sind entweder global definiert und gehören damit zu dem gesamten Prozess oder sie werden lokal innerhalb eines Unterblockes (*Scope*) definiert. Eine Variable besteht aus einem, innerhalb seines Blockes eindeutigen Namen und einem Datentyp. Der Typ muss laut Spezifikation [BPEL4WS, S.41] entweder ein WSDL Nachrichtentyp, ein simpler XMLSchema Datentyp oder ein XML Schema Element sein. Für die Definition der Variablen sieht der Standard unterhalb des `<process>` bzw. `<scope>`-Elementes das in Listing 10 angebenene XML-Element vor. Dabei ist von den optionalen Attributen, die den Datentyp festlegen genau eines zu verwenden.

```
<variables>
   <variable name="ncname"
             messageType="qname"? type=”qname”? element=”qname”?/>+
</variables>
```

Listing 10: BPEL-Syntax der Variablen Definition

Um die Definition einer Variable mit dem „ARIS SOA Designer“ zu modellieren, wird ein *„Variable“*-Element analog zur Spezifikation entweder mit dem *„ProcessStart”*- oder einem *„ScopeStart”*-Element verbunden. Diese Beziehung ist vom Typ *„is defined by”* bzw. *„defines”* (je nach Leserichtung). Um den Typ der Variable zu modellieren, wird noch ein entsprechendes Element, welches den Typ festlegt, mit dem *„Variable“*-Element verbunden. Diese

Elemente sind *"MessageType"*, *"XSDElement"* und *"XSDType"*, welche in der Abbildung 10 dargestellt sind.

Abbildung 10: Elementtypen für Datentypen einer Variable

Bei der Verwendung eines WSDL Nachrichtentyps kann der entsprechende Message Type selbst modelliert werden oder aus einer importierten WSDL-Datei stammen. Zur Modellierung eines neuen Nachrichtentypes wird das *„MessageType"*-Element zunächst mit den *„MessagePart"*-Elementen, welche die Teile der Nachricht spezifizieren, verbunden. Diese Elemente sind wiederum mit einem Datentyp verbunden, also mit einem *„XSD-Element"*- oder einem *„XSDType"*-Element. Ein auf diese Weise neu modellierter Nachrichtentyp wird beim BPEL-Export in der erzeugten WSDL-Datei, die den Prozess als Web Service beschreibt, angegeben. Soll ein importierter WSDL-Nachrichtentyp verwendet werden, benutzt man das entsprechende *„MessageType"*-Objekt, welches beim Import erzeugt wurde.

Soll der Datentyp einer Variable eines XML Schema Element sein, so muss dieses ebenso importiert sein (aus WSDL- oder XSD-Datei) und das dabei erzeugte *„XSDElement"* zur Definition einer Variablen verwendet werden. Eine Möglichkeit zur Definition durch einen XSD-Schema-Editor steht nicht zur Verfügung.

Dagegen wird ein simpler XML Datentyp zur Variablendefinition einfach durch ein *„XSD-Type"*-Element verwendet wird, welchem der Namen des entsprechenden Typs zugewiesen wurde.

Die Beispielmodellierung in Abbildung 11 zeigt die Definition globaler Variablen mit verschiedenen Typen. Der Typ der Variablen `'Variable1'` und `'Variable2'` ist dabei jeweils ein WSDL-Message Type. Allerdings wird der Typ von `'Variable1`, also `'ProcessMsgType'` im Beispiel neu modelliert. Er besteht aus den MessageParts `'Part1'` und `'Part2'`. Dagegen wurde der Typ `'Nachricht'` in einer importierten WSDL-Datei definiert und das erzeugte Objekt hier verwendet. Ebenso wurde der XML Schema Datentyp `'anyElement'` der Variable `'Variable3'` aus einer XML Schema Definition importiert. Die Variable `'Variable4'` ist dagegen von einem bekannten simplen XSD Datentyp. Hierbei genügt die Angabe des Namens, in diesem Fall `'float'` um ihn festzulegen.

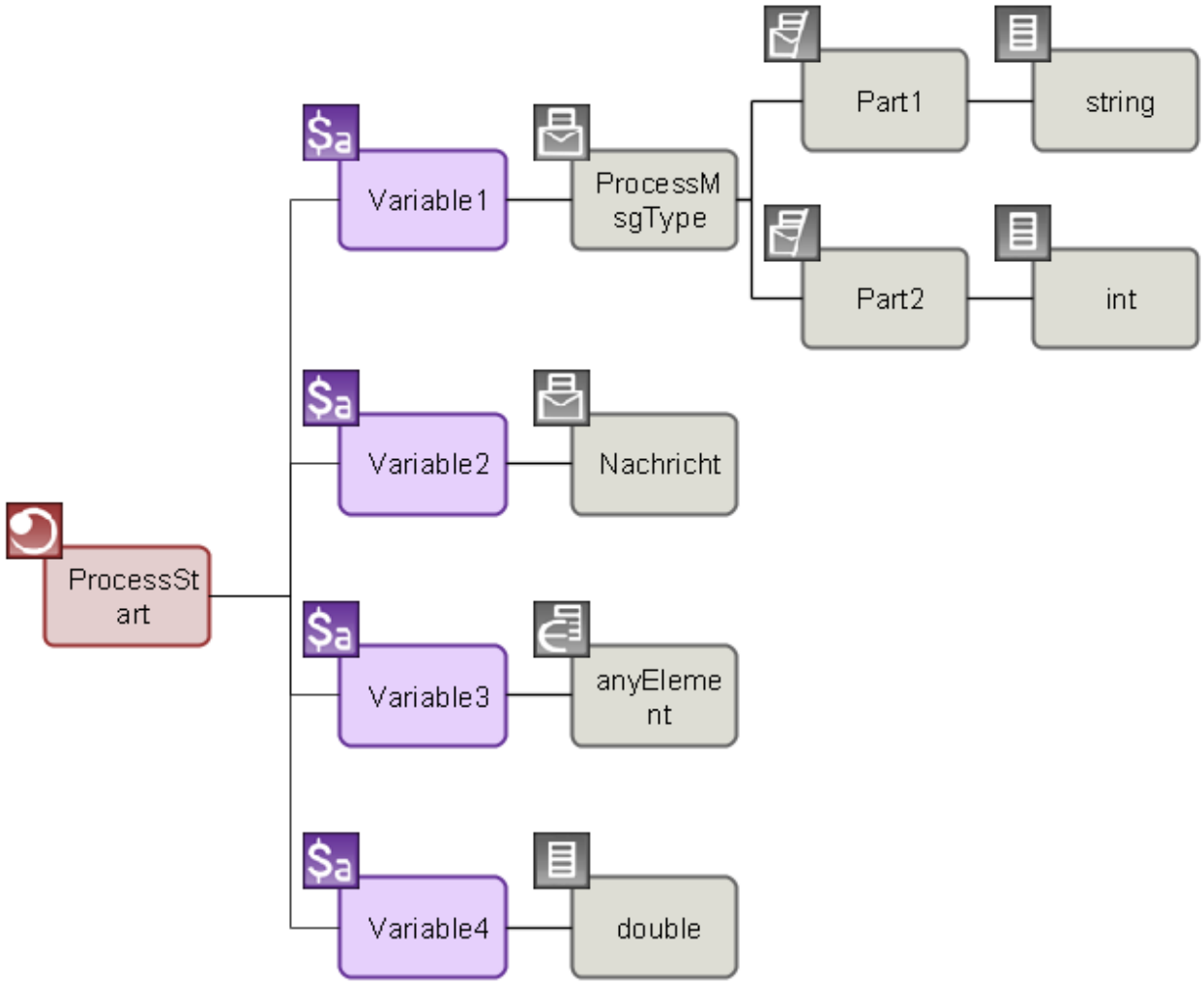

Abbildung 11: Beispielmodellierung einer Variablen Definition

Die für die Definition der Variablen nötigen Ausschnitte der daraus generierten BPEL-und WSDL-Dateien befinden sich in Listing 11.

```
<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         targetNamespace="http://www.ids-scheer.com/bpel"
         xmlns:tns="http://www.ids-scheer.com/bpel">

   <variables>
      <variable messageType="tns:ProcessMsgType" name="Variable1"/>
      <variable messageType="nsd1:Nachricht" name="Variable2"/>
      <variable xmlns:nsd2="http://exampleSchemaType"
                element="nsd2:anyElement" name="Variable3"/>
      <variable name="Variable4" type="xsd:double"/>
   </variables>
</process>
```

aus **beispielprozess.bpel**

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:nsd1="http://beispiel"
             xmlns:nsd2="http://exampleSchemaType"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             targetNamespace="http://www.ids-scheer.com/bpel">

   <message name="ProcessMsgType">
      <part name="Part1" type="xsd:string"/>
      <part name="Part2" type="xsd:int"/>
   </message>
</definitions>
aus beispielprozess.bpel
```

Listing 11: Aus Beispielmodell generierte Variablen und Message Definition

Der BPEL-Prozess enthält folglich die Definition der modellierten Variablen, mit den entsprechend Namen und Datentypen. Dabei ist der Namensraum des modellierten Nachrichtentyps (`'tns'`) der Zielnamensraum des Prozesses. Auf diesen verweist auch die WSDL-Beschreibung, die eine korrekte Definition des modellierten Nachrichtentyps `'ProcessMsgType'` beinhaltet. Die Namensräume der importierten Typen (`'nsd1'` und `'nsd2'`) wurden in den importierten Dateien so festgelegt und hier richtig übernommen. Damit wird die BPEL-Spezifikation zur Definition von Variablen erfüllt und die Modellierung riechtig umgesetzt.

3.5 Message Properties und Correlation Sets

Message Properties repräsentieren bestimmte Teile von Nachrichten und geben Datentypen einen Namen, der aussagekräftiger ist, als der Typ selbst. Diese Eigenschaften werden in BPEL innerhalb so genannter *Correlation Sets* verwendet, welche die Zuordnung von Nachrichten zu einzelnen Prozessinstanzen über den Inhalts der Nachrichten ermöglichen.

3.5.1 Message Property Definition

Die Definition einer Message Property gibt einem simplen XML-Schema-Datentyp einen eindeutigen Namen. Um eine Eigenschaft aus einem Teil (Message Part) eines Nachrichtentyps zu addressieren, dient ein `propertyAlias`. Hierbei wird der entsprechende Teil der Nachricht festgelegt und durch eine optionale Anfrage daraus das gesuchte Element heraus gefiltert. Die in der BPEL-Spezifikation vorgesehene Syntax (Listing 12) für eine Property bzw. PropertyAlias Definition sind folgende neue WSDL-Definitionen. [BPEL4WS, S. 37f.]

```
<definitions name="ncname"
     xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/">

  <bpws:property name="ncname" type="qname"/>
```

```
    <bpws:propertyAlias propertyName="qname"
              messageType="qname" part="ncname" query="queryString"/>
</wsdl:definitions>
```

Listing 12: WSDL-Syntax zur Definition von Message Properties

Der „ARIS SOA Designer“ bietet zur Modellierung einer Message Property ein entsprechendes *„Property“*-Element. Dieses wird zur Definition einer Eigenschaft mit einem *„XSD-Type“*-Element verbunden. Um einen PropertyAlias zu definieren, wird das *„Property“*-Element analog zur oben angebenen Spezifikation zusätzlich noch mit einem *„MessageType“*- und einem *„MessagePart“*-Element verbunden. Das Attribut `query` des `<propertyAlias>`-Elementes muss über das als Attribut *„Query expression“* der Verbindung zwischen dem *„Property“*- und dem *„MessagePart“*-Element gesetzt werden.

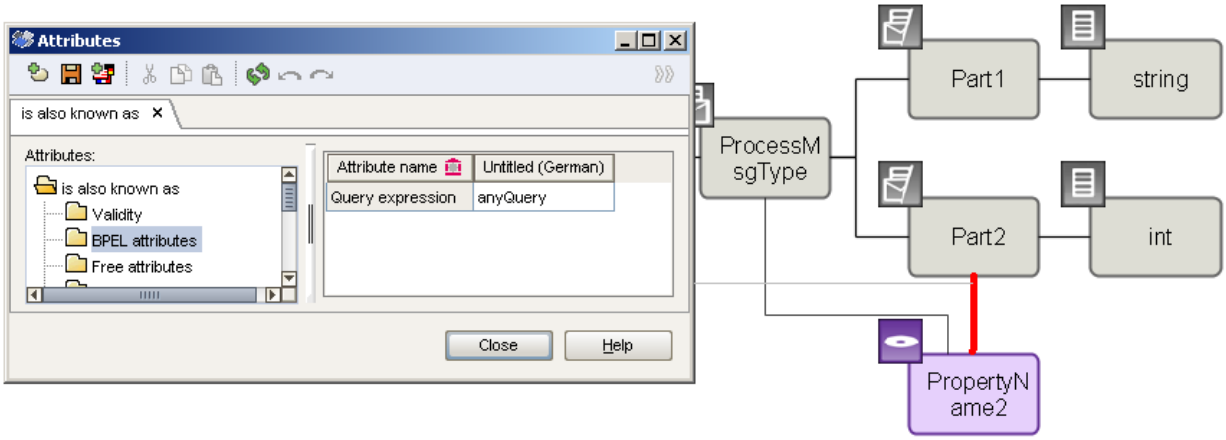

Abbildung 12: Beispielmodellierung einer PropertyAlias Definition

In Abbildung 12 ist solch eine Modellierung einer PropertyAlias Definition dargestellt. Diese besitzt die Pseudo-Anfrage `'anyQuery'`. Die BPEL-Export-Funktion erzeugt dabei folgende PropertyAlias Definition:

```
<bpws:propertyAlias messageType="tns:ProcessMessageType" part="Part2"
      propertyName="tns:PropertyName2" query="anyQuery"/>
```

Listing 13: Aus Beispielmodell generierte propertyAlias Definition

Da die Property-Definitionen eine Erweiterung des WSDL-Standards sind, werden so modellierte Property-Definition beim BPEL-Export in der WSDL-Datei, die den gesamten Prozess beschreibt, angegeben. Allerdings nur dann, wenn die modellierte Property auch in mindestens einem Correlation Set verwendet wird. Deren Modellierung wird nachfolgend beschrieben.

Ein Beispiel zur Modellierung von Definition von Propertys befindet sich in Abbildung 13 und wird im Rahmen des nächsten Abschnittes erläutert.

3.5.2 Definition von Correlation Sets

Ein *Correlation Set* ist eine Menge von zusammengehörenden Properties. Es wird verwendet, um die Kommunikation mit anderen Web Services zu unterstützen. Dabei hält es Nachrichten, die für dieselbe Kommunikation mit einer Prozessinstanz genutzt werden zusammen. Die Definition eines Correlation Sets (Listing 14) gruppiert eine Menge von Message Properties und gibt dieser Menge einen eindeutigen Namen [BPEL4WS, S. 47f]. Die Definition erfolgt entweder innerhalb des gesamten Prozesses oder innerhalb eines Unterblockes.

```
<correlationSets>?
   <correlationSet name="ncname" properties="qname-list"/>+
</correlationSets>
```

Listing 14: BPEL-Syntax der Correlation Set Definition

Analog zu der Spezifikation erfolgt auch die Modellierung eines Correlation Sets mit dem „ARIS SOA-Designer“. Ein entsprechendes *„CorrelationSet“*-Element wird einfach mit den *„Property“*–Elementen verbunden, welche Teil des Correlation Sets sein sollen. Zur Definition wird dieses *„CorrelationSet“*-Element entweder mit einem *„ProcessStart“*- oder einem *„ScopeStart“*-Element verbunden. Damit können globale oder lokalen Correlation Sets definiert werden.

Die Abbildung 13 zeigt eine Beispielmodellierung von Message Properties und einem Correlations Set, welches diese verwendet. Die beiden Properties `'KundenNr'` und `'BestellNr'`

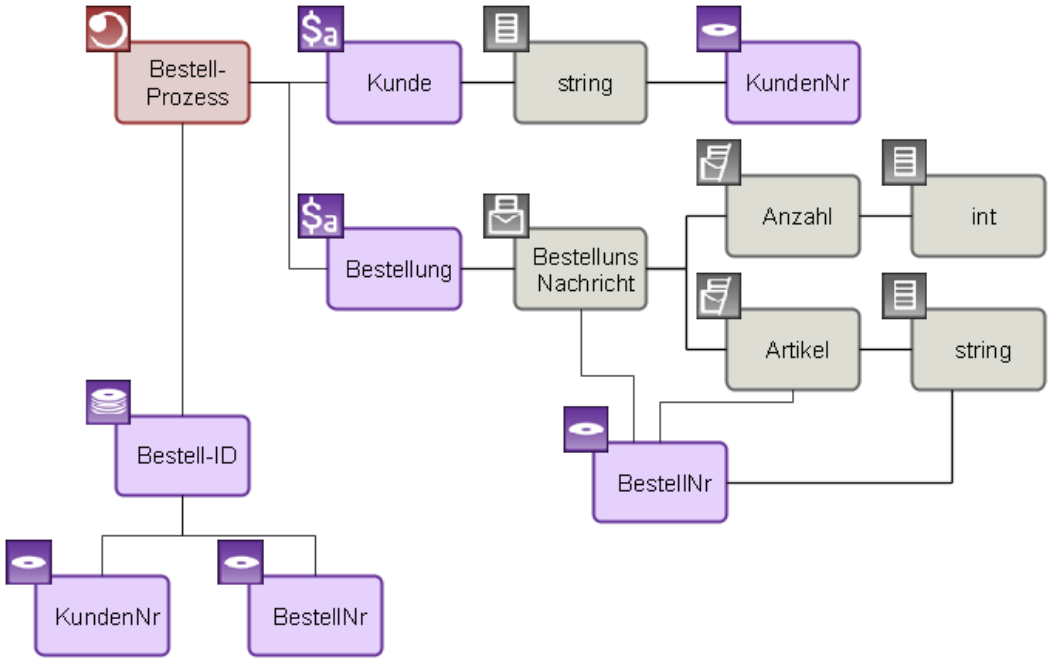

Abbildung 13: Beispielmodellierung einer Property und CorrelationSet Definition

wurden zunächst durch Verbinden zu den entsprechenden Datentypen definiert. Dabei ist `'BestellNr'` eine zuvor schon erläuterte Modellierung (Abb. 12) einer *„popertyAlias“* Definition. Anschließend wurden die so erzeugten *„Property“*-Objekte dem Correlation Set `'Bestell-Id'` hinzugefügt.

Die relevanten Ausschnitte des daraus generierten BPEL- bzw. WSDL-Codes sind im Listing 15 angegeben. Die BPEL-Datei enthält dabei das Correlation Set, welches die beiden hinzugefügten Properties `'tns:KundenNr' 'tns:BestellNr'` verwendet. Der Namensraumprefix `'tns'` verweist dabei auf denselben Namensraum, in den aus die WSDL-Beschreibung mit den Property Definitionen verweist. Die BPEL-Spezifikation wurde also korrekt umgesetzt.

```
<correlationSets>
   <correlationSet name="Bestell-ID"
                   properties="tns:KundenNr tns:BestellNr"/>
</correlationSets>
```

```
<bpws:property name="KundenNr" type="xsd:string"/>
<bpws:property name="BestellNr" type="xsd:string"/>
<bpws:propertyAlias messageType="tns:BestellunsNachricht" part="Part2"
                    propertyName="tns:BestellNr" query="anyQuery"/>
```

Listing 15: Aus Beispielmodell generierte Correlation Set und Propertiy Definitionen

3.6 Einfache Aktivitäten

Unter den einfachen Aktivitäten, versteht man in BPEL solche Aktivitäten, die nicht aus anderen aufgebaut sind. Sie sind sozusagen atomar. Dies sind der Aufruf einer Operation eines Web Services (*Invoke*), das Anbieten der Schnittstelle für den Prozess als Web Service (*Receive* und *Reply*) und das Ändern von Werten der Variablen (A*ssign*). Zudem gibt es noch Aktivitäten, die einen Fehler signalisieren (*Throw*), eine bestimmte Zeit warten (*Wait*) oder keine Operation ausführen (*empty*). Der „ARIS SOA Designer“ bietet für alle diese Aktivitäten ein entsprechendes Element. Diese sind in der Abbildung 14 dargestellt und werden in den nachfolgenden Abschnitten erläutert.

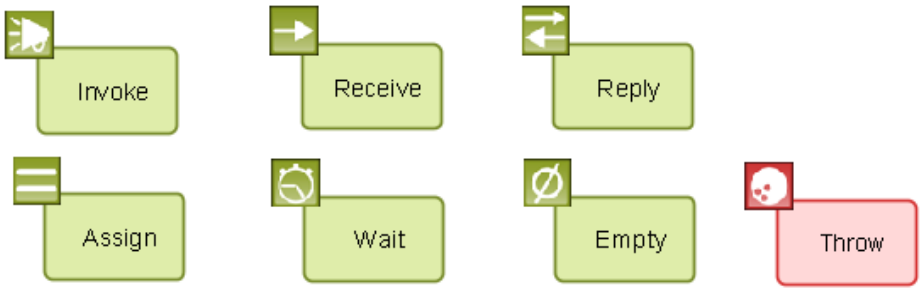

Abbildung 14: Elementtypen für „einfache Aktivitäten“

3.6.1 Standardattribute und -elemente

Die Spezifikation [BPEL4WS, S. 53] sieht den grundsätzlichen Aufbau einer Aktivität wie in Listing 16 dargestellt vor. Jede Aktivität besitzt als optionale Attribute einen Namen sowie die Attribute `'joinCondition'` und `'suppressJoinFailure'`. Wobei diese beide im Zusammenhang mit der *Flow-Aktivität* von Bedeutung sind. Ebenfalls in diesem Kontext werden die so genannten Standardelemente `<source>` und `<target>` benutzt. Auf die genauen Bedeutungen dieser Attribute und Elemente wird nicht eingegangen, sondern auf die Spezifikation verwiesen. Eine kurze Beschreibung findet sich jedoch im Abschnitt „3.7.5 Flow Aktivität“.

```
<ACTIVITY standard-attributes ...>
   standard-elements
   ...
</ACTIVITY>
```

wobei für `standard-attributes` gilt:

```
name="ncname"?
joinCondition="bool-expr"?
suppressJoinFailure="yes|no"?>
```

und für `standard-attributes` gilt:

```
<source linkName="ncname" transitionCondition="bool-expr"?/>*
<target linkName="ncname"/>*
```

Listing 16: Standardattribute und -elemente einer BPEL Aktivität

Bei der Modellierung mit dem „ARIS SOA Designer“ besitzt jedes Element einen Namen, welcher im oben genannten Zusammenhang verwendet wird. Die weiteren Attribute `joinCondition` und `suppressJoinFailure` lassen sich über die gleichnamigen Attribute aller Elemente, die zur Modellierung Aktivität dienen, festlegen. Die BPEL-Elemente `<source>` und `<target>` lassen sich über *„Links“* im Zusammenhang mit einer *Flow-Aktivität* modellieren. Näheres dazu wird in dem entsprechenden Abschnitt über Flows erläutert.

3.6.2 Aufruf von Operationen eines Web Service (Invoke)

Um Web Services über einen Partner Link aufzurufen, wird die *Invoke-Aktivität* verwendet. Der Aufruf kann entweder synchron oder asynchron erfolgen. Die Syntax dafür ist dieselbe (Listing 17). Sie unterscheidet sich lediglich in der Nutzung der optionalen Attribute [BPEL4WS, S.54f]. Eine Invoke Aktivität benötigt einen Partner Link, über welchen mit dem Web Service kommuniziert wird, sowie die dazugehörigen Porttypen und die aufzurufende Operation. Zusätzlich besitzt sie noch Variablen, die den Parametern der aufgerufenen Operation entsprechen. Hierbei wird eine `inputVariable` nur bei einem synchronen Aufruf benötigt. Außerdem können zu einer Invoke-Aktivität noch Correlation Sets, sowie Fehler- und Kompensationsbehandlung hinzugefügt werden.

```
<invoke partnerLink="ncname" portType="qname" operation="ncname"
        inputVariable="ncname"? outputVariable="ncname"?
        standard-attributes>
   standard-elements
   <correlations>?
      <correlation set="ncname" initiate="yes|no"?
                   pattern="in|out|out-in"/>+
   </correlations>
   <catch faultName="qname" faultVariable="ncname"?>*
      activity
   </catch>
   <catchAll>?
      activity
   </catchAll>
   <compensationHandler>?
      activity
   </compensationHandler>
</invoke>
```

Listing 17: BPEL-Syntax der Invoke Aktivität

Die Modellierung einer solchen Aktivität wird anhand des Beispieles in Abbildung 21 erläutert. Darin wird eine Invoke Aktivität `'BestellungAusfuehren'` modelliert. Das entsprechende *„Invoke“*-Element wird dazu analog zur Spezifikation mit einem *„ParterLink“*- und einem *„Operation“*-Element verbunden. Sind diese korrekt definiert, kann der „ARIS SOA Designer“ daraus den entsprechenden Porttyp ermitteln. Dies reicht prinzipiell schon aus um eine Invoke Aktivität zu modellieren. Da weiteren Konstrukte (vgl. Listing 17) optional sind, könnten sie auch weggelassen. Um jedoch ihre Modellierung darzustellen werden sie im Beispiel verwendet. So wird das *„Invoke“*-Element auch noch mit den *„Variable“*-Elementen `'BestelleNachricht'` und `'Bestaetigung'` verbunden. Der Beziehungstyp ist dabei entweder *„hasInput“* oder *„hasOutput“*. Dies wird auch durch die Pfeilrichtung angezeigt. Um ein Correlation Set zuzuweisen wird das Objekt `'BestellID'`, welches zuvor definiert sein sollten, ebenfalls mit der Invoke-Aktivität verbunden. Die Attribute `initiate` und `pattern` des in diesem Zusammenhang verwendeteten Correlation Sets können über die gleichnamigen Attribute der Kante zwischen dem *„CorrelationSet“*- und dem *„Invoke“*-Element ausgewählt werden. Im Beispiel wurde für das Attribut *„Pattern“* nichts ausgewählt und *„Initiate“* auf `'nein'` gesetzt. Der optionale CompensationHandler wird modelliert, indem die Aktivität, welche zur Kompensation ausgeführt werden soll, direkt mit dem „Invoke“-Element über eine *„defines Compensation“*-Beziehung verbunden ist. In diesem Fall ist das die Aktivität `'BestellungStorno'`. Schließlich wird auch noch die Fehlerbehandlung modelliert. Dabei wird im Beispiel der Fehler `'nichtLieferbar'` aufgefangen und zur Behandlung die Aktivität `'sendeNachricht'` ausgeführt Auf eine zusätzliche Fehlervariable wurde verzichtet. Außerdem ist die Aktivität `'empty'` über eine *„catches all“*-Beziehung mit dem „Invoke“-Element verbunden. Dies bedeutet, dass alle weiteren Fehler aufgefangen und durch die entsprechende Aktiviät `'empty'` behandelt werden.

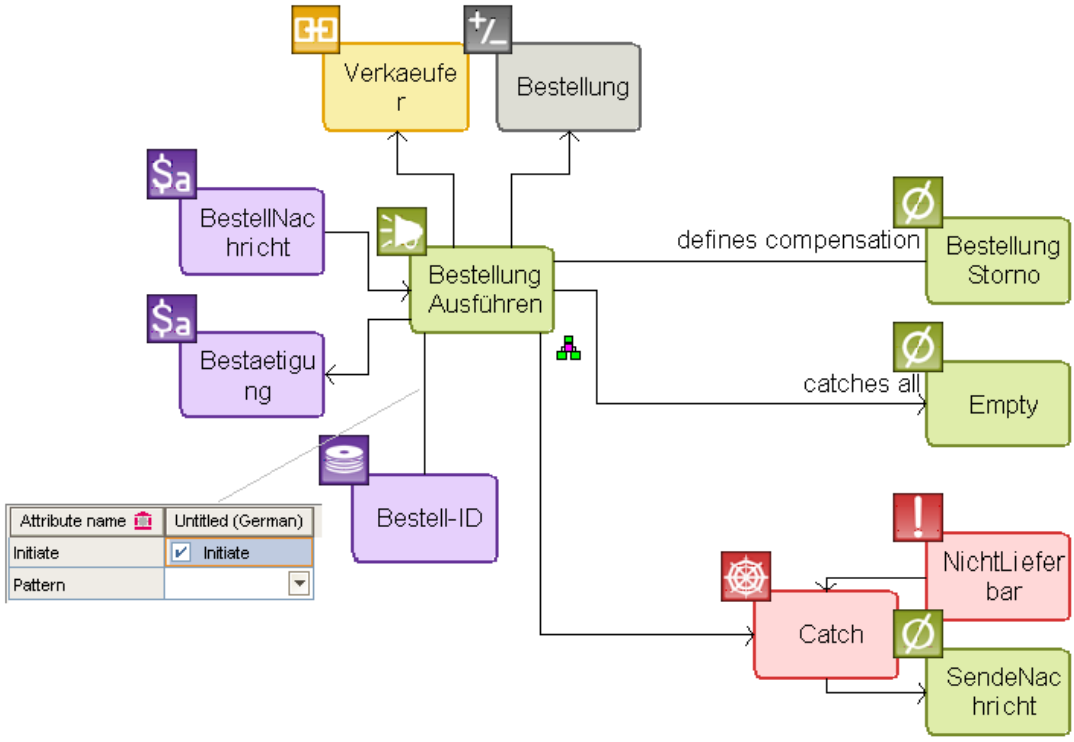

Abbildung 15: Beispielmodellierung einer Invoke Aktivität

In Listing 18 ist der Ausschnitt dieser Invoke Aktivität, welche die BPEL-Importfunktion aus zugehörigem Prozess generiert hat, angegeben. Dabei wird der PortType `'tns:ShopPT'` aus den definierten Partner Links richtig erkannt. Vorraussetzung hierfür ist, dass die Definition des Partner Links richtig umgesetzt wurde und der entsprechende PartnerLinkTyp darin enthalten ist. Auch die Fault- und Compensation Handler wurden korrekt erzeugt.

```
<invoke name="BestellungAusführen"
      partnerLink="Verkaeufer" portType="tns:ShopPT" operation="Bestellung"
      inputVariable="BestellNachricht" outputVariable="Bestaetigung">
   <correlations>
      <correlation set="Bestell-ID"/>
   </correlations>
  <catch faultName="tns:NichtLieferbar">
    <empty name="SendeNachricht"/>
  </catch>
  <catchAll>
    <empty name="Empty"/>
  </catchAll>
  <compensationHandler>
    <empty name="BestellungStorno"/>
  </compensationHandler>
</invoke>
```

Listing 18: Aus Beispielmodell generierte Invoke Aktivität

Zu beachten ist allerdings das erzeugte Correlation Set `<correlation set="Bestell-ID"/>`. Das entspricht zwar der Modellierung, aber nicht der Spezifikation. Diese sieht nämlich wie folgt aus: `<correlation set="ncname" initiate="yes|no"? pattern="in|out|out-in"/>`

(vgl. Listing 17). Dies ist der Default-Wert und kann deshalb weggelassen werden. Allerdings ist das fehlende Attribut `pattern` nicht optional. Da es zwar in der Modellierung nicht gesetzt wurde, entspricht das erzeugte Correlation Set dem Modell. Jedoch wird durch diese fehlerhafte Modellierung syntaktisch fehlerhafter BPEL-Code generiert, ohne dass eine beim Export Warnmeldung ausgegeben wird.

3.6.3 Web Service Operationen anbieten (Receive/Reply)

Ein Geschäftsprozess stellt seine Dienste hauptsächlich über *Receive Aktivitäten* und dazugehörigen *Reply-Aktiviäten* zur Verfügung. Die Syntax einer Receive Aktivität ist in Listing 19 angegeben. Die Aktivität gibt den Partner Link an, über den eine Nachricht empfangen wird und die Operation, die aufgerufen werden soll. Zusätzlich kann noch eine Variable hinzugefügt werden, in der die empfangene Nachricht gespeichert wird, sowie ein Correlation Set zur Identifikation kommunizierender Prozessinstanzen. Die Receive Aktivität kann außerdem dazu genutzt werden um beim Empfangen einer Nachricht eine Prozessinstanz des BPEL-Prozesses zu erzeugen. Dazu wird das Attribut `createInstance` verwendet [BPEL4WS, S. 55ff].

```
<receive partnerLink="ncname" portType="qname" operation="ncname"
         variable="ncname"?
         createInstance="yes|no"?
         Standard-attributes>
   standard-elements
   <correlations>?
      <correlation set="ncname" initiate="yes|no"?>+
   </correlations>
</receive>
```

Listing 19: BPEL-Syntax der Receive Aktivität

Zur Modellierung wird das *„Receive“*-Element wird mit einem *„PartnerLink“*- und einem *„Operation“*-Element verbunden. Vorausgesetzt die Partner Links wurden zuvor korrekt definiert, erkennt der „ARIS SOA Designer“ daraus den entsprechenden Porttyp. Das Attribut `createInstance` der Receive-Aktivität kann über das gleichnamige Attribut eines *„Receive“*-Elementes ausgewählt werden. Optional kann noch eine Correlation Set hinzugefügt werden. Dessen Attribut `initiate` lässt sich über entsprechendes Attribut der Beziehung zwischen *„Receive“*- und *„Correlation Set“*-Element setzen.

In der Abbildung 16 wurde eine beispielhafte Receive Aktivität `'Anfrage-erhalten'` modelliert. Sie erwartet eine Nachricht über den Partner Link `'Client'`, welche die Operation `'ExecuteAnfrage-erhalten'` aufruft. Die Aktivität soll zur Erzeugung einer Prozessinstanz genutzt dienen. Dazu wird das Attribut *„Create Instance“* des `'Anfrage-erhalten'`-Objektes ausgewählt. Ein Correlation Set `'CorrelationSet'` ist ebenfalls der Aktivität zugefügt, und dessen Attribut *„Initiate“* ist gesetzt wurde.

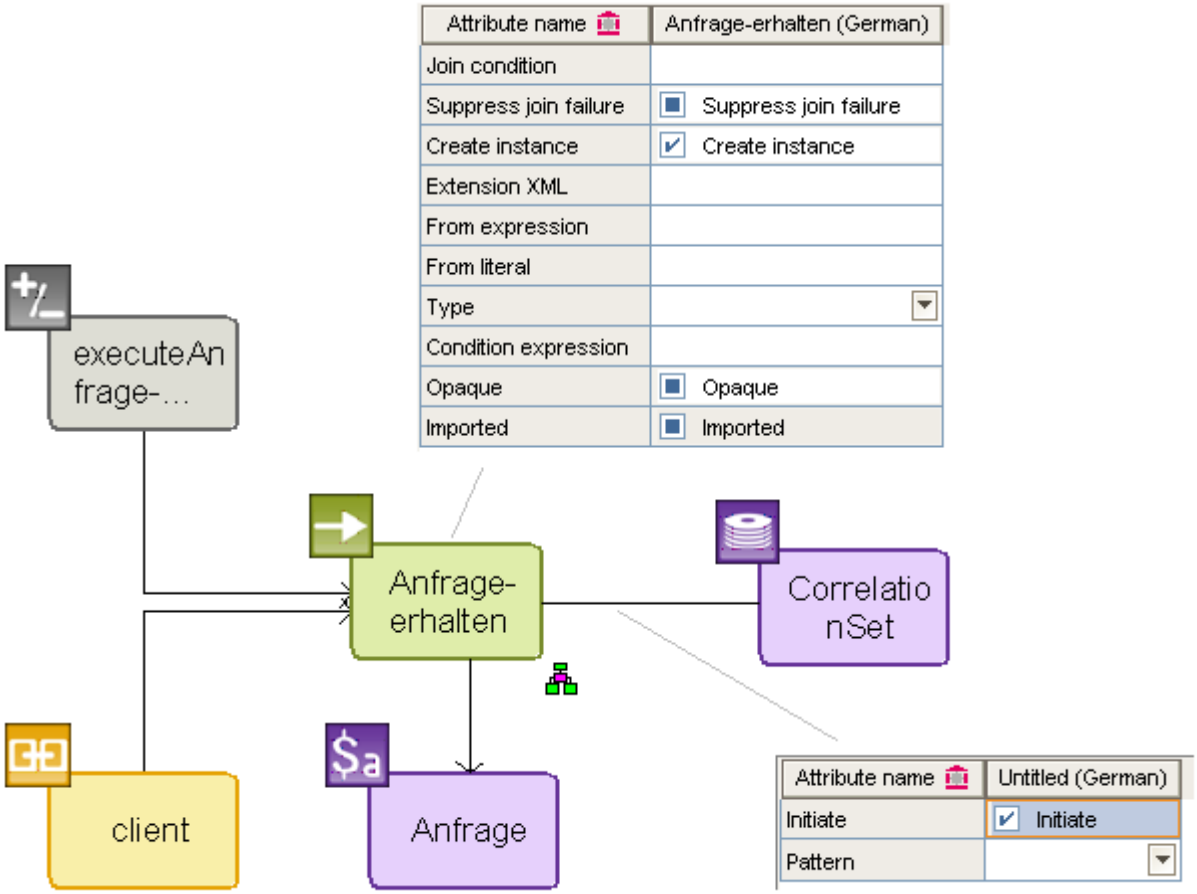

Abbildung 16: Beispielmodellierung einer Receive Aktivität

Das aus diesem Modell erzeugte BPEL-Element ist in Listing 20 abgebildet. Die Attribute des `<receive>`-Elementes sind entsprechend der Modellierung erzeugt wurden. Der entsprechende PortType `"tns:ClientPT"`, welcher in der Modellierung des Aktivität nicht vorkommt, wurde korrekt aus der hier nicht abgebildeten Partner Link Definition übernommen. Die Attribute `variable` und `createInstance` wurden ebenfalls der Modellierung entsprechend generiert. Gleiches gilt für das verwendete Correlation Set und dessen Attribute `initiate`.

```
<receive name="Anfrage-erhalten"
         createInstance="yes"
         operation="executeAnfrage-erhalten"
         partnerLink="client" portType="tns:ClientPT"
         variable="Anfrage">
   <correlations>
      <correlation initiate="yes" set="CorrelationSet"/>
   </correlations>
</receive>
```

Listing 20: Aus Beispielmodell generierte Receive Aktivität

Die Exportfunktion liefert also für die modllierte Receive -Aktivität BPEL-Code, welcher die Spezifikation erfüllt.

Um zu einer vorangegangen, über eine Receive Aktivität erhaltene Anfrage, eine Antwort zu senden, dient die *Reply Aktivität*. Deren Syntax ist in Listing 21 abgebildet und ist ähnlich zu der Receive-Aktivität. Die Reply-Aktivität gibt den Partner Link über den gesendet wird und die aufzurufende Operation an. Die zu sendende Nachricht ist in einer Variable der Aktivität zugewiesen. Eine Verwendung von Correlation Sets um eine Beziehung zwischen Anfrage und Antwort herzustellen, ist ebenso möglich. Beim Auftreten eines Fehlers wird der über `faultName` spezifizierte Fehler geworfen. [BPEL4WS, 55ff].

```
<reply partnerLink="ncname" portType="qname" operation="ncname"
       variable="ncname"? faultName="qname"?
       Standard-attributes>
   standard-elements
   <correlations>?
      <correlation set="ncname" initiate="yes|no"?>+
   </correlations>
</reply>
```

Listing 21: BPEL-Syntax der Reply Aktivität

In Abbildung 17 ist eine beispielhafte Modellierung einer Reply Aktivität dargestellt. Diese ist prinzipiell gleich mit der zuvor diskutierten Modellierung eines Receive (Abb. 16). Die Reply-Aktivität `'Anfrage-erhalten'` wird mit den zugehörigen *„PartnerLink“*- und *„Operation“*-Objekten verbunden. Zusätzlich wird der Aktivität noch eine Variable `'BestellNr'`, eine Fehler `'Fault'` und ein Correlation Set `'CorrelationSet'` zugewiesen.

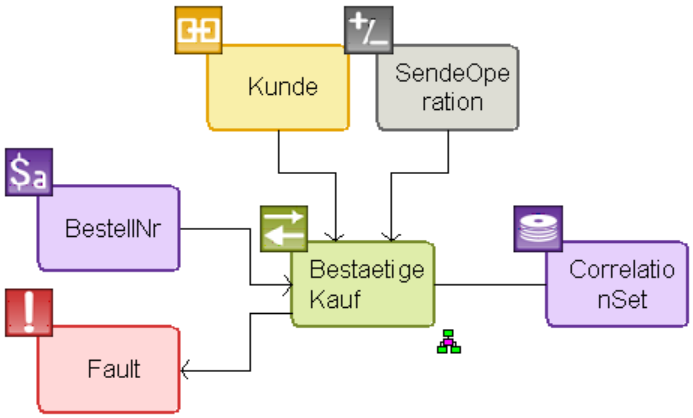

Abbildung 17: Beispielmodellierung einer Reply Aktivität

Die aus diesem Modell generierte `<reply>`-Aktivität ist in Listing 22 angegeben. Bei einer richtig erfolgten Partner Link Definition, werden das entsprechende PortTyp-Attribute korrekt erzeugt. Die Variable, der Fehler sowie das Correlation Set werden ebenfalls dem Modell entsprechend generiert. Dabei ist zu beachten, dass das Attribut *„Initiate“* der Beziehung `'BestaetigeKauf' - 'CorrelationSet'` auf 'nein' gesetzt wurde. Dementsprechend wird das Attribut `initiate` nicht angezeigt, da dies dem Default-Wert entspricht.

```
<reply name="BestaetigeKauf"
   partnerLink="Kunde" operation="SendeOperation" portType="tns:ClientPT"
```

```
   faultName="tns:Fault" variable="BestellNr">
  <correlations>
    <correlation set="CorrelationSet"/>
  </correlations>
</reply>
```

Listing 22: Aus Beispielmodell generierte Reply-Aktivität

Die Modellierung der Receive und Reply-Aktivitäten liefert, sofern die Partner Links richtig erzeugt wurde, der Spezifikation entsprechenden BPEL-Code.

3.6.4 Wertzuweisungen

Ein Zuweisung über die *Assign-Aktivität*, dient dazu den Zustand einer Variablen zu ändern. Dazu können entweder Werte von Variablen oder Endpunktreferenzen kopiert werden oder über das `expression` Attribut Berechnungen auf Nachrichteninhalten durchgeführt werden. Die Syntax dieser Aktivität ist in Listing 23 abgebildet [BPEL4WS, S.42f].

```
<assign standard-attributes>
  standard-elements
  <copy>+
    from-spec
    to-spec
  </copy>
</assign>
```

Wobei `from-spec` aus:

```
<from variable="ncname" part="ncname"?/>
<from partnerLink="ncname" endpointReference="myRole|partnerRole"/>
<from variable="ncname" property="qname"/>
<from expression="general-expr"/>
<from> ... literal value ... </from>
```

und `from-spec` aus:

```
<to variable="ncname" part="ncname"?/>
<to partnerLink="ncname"/>
<to variable="ncname" property="qname"/>
```

Listing 23: BPEL-Syntax der Assign Aktivität und zugehörige Copy-Statements

Der „ARIS SOA Designer“ bietet zur Modellierung ein *„Assign“*- und ein *„Copy“*-Element. Das *„Assign“*-Element wird an der gewünschten Stelle als Aktivität in den Prozess eingefügt. Es darf der Spezifikation entsprechend mit beliebig vielen *„Copy“*-Elementen verbunden sein. Mit einem *„Copy“*-Element können nun die gewünschten Quell bzw. Zielelemente verbunden werden. Dies sind, dem Listing 23 entsprechend, die Elemente *„Variable“*, *„MessagePart“*, *„Property“* oder *„PartnerLink“*, welche in einem jeweiligen `<from>`- bzw. `<to>`-Statements verwendet werden. Dabei spielt natürlich die Richtung der Verbindung eine bedeutende Rolle. Diese wird durch einen Pfeil angezeigt, der angibt, ob der Beziehungstyp *„to“* oder *„from“* ist. Beim Kopieren eines Partner Links, wird das Attribut `endpointReference` über das Attribut *„Role Type“* der *„from“*-Beziehung gewählt. Bei `<from>`-

Statements, die eine Expression oder ein Literalwert verwenden, erfolgt die Modellierung über entsprechende Attribute des *„Copy“*-Elementes. Diese sind *„From Expression“* bzw. *„From Literal“*.

Die Abbildung 18 enthält eine beispielhafte Modellierung einer Assign-Aktivität, mit drei Copy-Anweisungen. Die Anweisung `'Copy1'` realisiert das jeweils erste `<from>`- bzw. `<to>`-Statement. Hierbei wird von einem Teil `'MsgPart1'` des MessagesTypes einer Variable `'Variable1'` in eine andere Variable `'Variable2'` kopiert. Bei der Zielvariable wird auf den optionalen MessagePart verzichtet.

Das zweite Copy-Statement `'Copy2'` kopiert von einem PartnerLink `'CustomerPL'` in einen anderen, welcher dem Namen `'ServicePL'` besitz. Es spiegelt also jeweils die zweite Variante der `from-spec` und `to-spec` wieder. Dabei ist über das Attribut *„Role Type“* der *„from“*-Beziehung noch die `EndpointReference` auszuwählen, in disem Fall wurde `PartnerRole` gewählt.

Beim dritten Copy wird die vierte Variante der `from-spec` umgesetzt. Um `expression` mit einem Wert zu belegen, wird das Attribut *„From expression“* des Objektes `'Copy3'` mit dem Ausdruck `'anyFromExpression'` belegt. Das Ziel erfüllt diesmal die dritte `to-spec` -Variante. Es wurde dazu ein *„Property“*-Objekt `'Property'` und eine Variable `'Variable2'` verwendet.

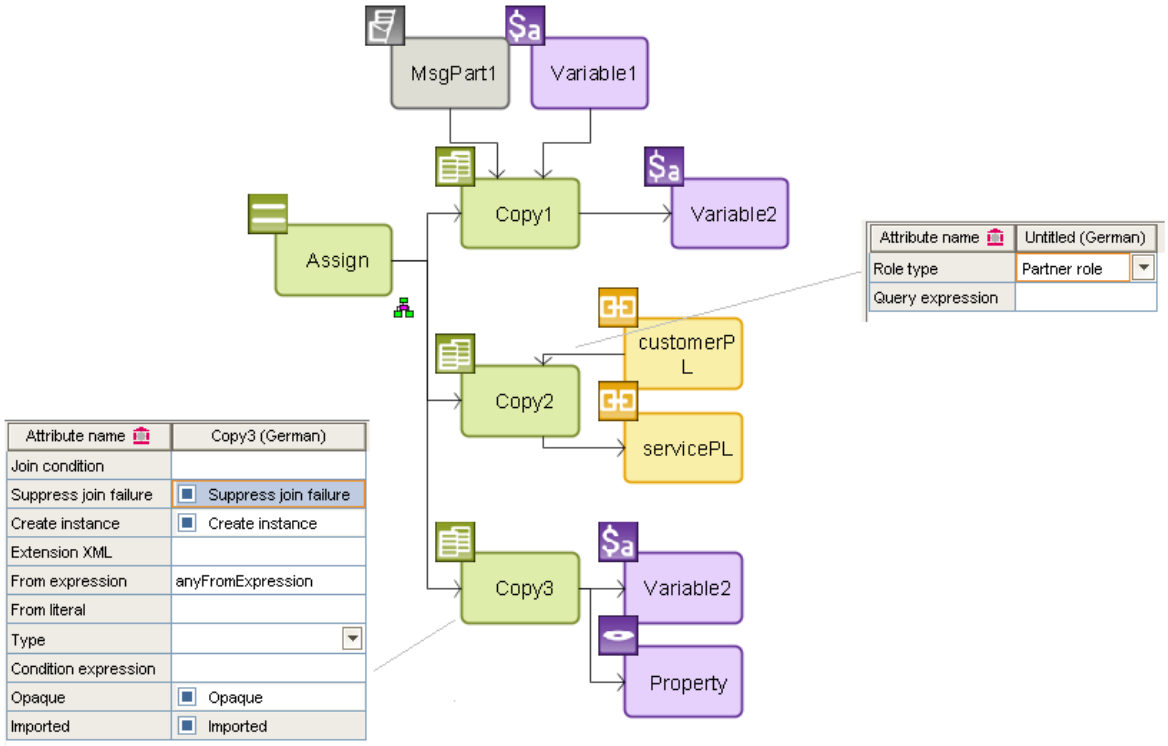

Abbildung 18: Beispielmodellierung einer Assign Aktivität

Aus dem obigen Beispiel wird mittels der Exportfunktion folgender BPEL-Code (Listing 24) generiert. Dieser entspricht der Spezifikation und der Modellierung.

```
<assign name="Assign">
  <copy>
    <from part="MsgPart1" variable="Variable1"/>
    <to variable="Variable2"/>
  </copy>
  <copy>
    <from endPointReference="partnerRole" partnerLink="customerPL"/>
    <to partnerLink="servicePL"/>
  </copy>
  <copy>
    <from expression="anyFromExpression"/>
    <to property="Property" variable="Variable2"/>
  </copy>
</assign>
```

Listing 24: Aus Beispielmodell generierte Assign Aktivität

3.6.5 Fehler signalisieren

Die *Throw Aktivität* signalisiert einen internen Fehler im Prozess. Sie „wirft“ einen Fehler, welcher mit Hilfe eines *FaultHandlers* aufgefangen und behandelt werden kann. Um einen Fehler zu signalisieren, gibt die Throw Aktivität den Namen des Fehlers und optional dazu noch weiter Informationen in einer Fehlervariable aus. [BPEL4WS, S. 57] Die Syntax sieht folgendermaßen aus:

```
<throw faultName="qname" faultVariable="ncname"? Standard-attributes>
   standard-elements
</throw>
```

Listing 25: BPEL-Syntax der Throw Aktivität

Der „ARIS SOA Designer“ bietet für diese Aktivität eine *„Throw“*-Element. Dieses wird, wie in Abbildung 19 dargestellt, entsprechend der BPEL-Spezifikation mit einem Fault-Element und optional noch mit einer Variable-Element verbunden.

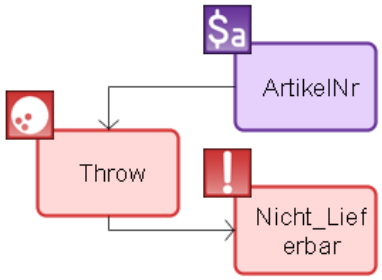

Abbildung 19: Beispielmodellierung einer Throw Aktivität

Die obige Beispielmodellierung einer Throw-Aktivität liefert beim Export folgenden spezifikationsgetreuen BPEL-Code.

```
<throw name="Throw"
      faultName="tns:Nicht_Lieferbar"
      faultVariable="ArtikelNr"/>
```

Listing 26: Aus Beispielmodell generierte Throw Aktivität

3.6.6 Warten

Die *Wait Aktivität* verzögert einen Prozess für eine bestimmte Zeitspanne oder bis zu einem bestimmten Zeitpunkt [BPEL4WS, S.57]. Das Listing 27 zeigt die Syntax dieser Aktivität.

```
<wait (for="duration-expr" | until="deadline-expr") standard-attributes>
   standard-elements
</wait>
```

Listing 27: BPEL Syntax der Wait Aktivität

Um diese Aktivität zu modellieren, gibt es im ARIS SOA Designer ein *„Wait“*-Element. Mit Hilfe der Attribute dieses Elementes wird festgelegt, wie lange gewartet werden soll. Über das Attribut *„Type“* wird ausgewählt, ob eine bestimmte Zeit (`Type=for`) oder bis zu einem Zeitpunkt (`type=until`) gewartet wird. Der Wert wird durch das Attribut *„Condition expression“* festgelegt. In der Abbildung 20 wird eine solche Wait-Aktivität modelliert, die bis zum 24.Dezember 2008 wartet und einen Prozess verzögert.

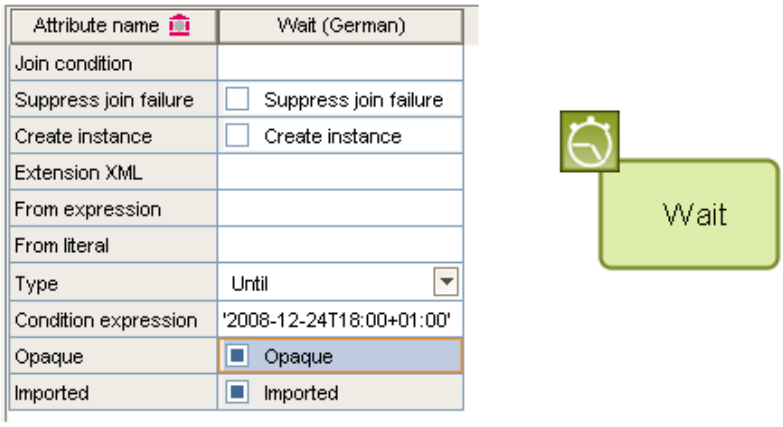

Abbildung 20: Beispielmodellierung einer Wait Aktivität

Aus diesem *Wait*-Element wird beim Export folgender Code erzeugt, welcher der Spezifikation entspricht.

```
<wait name="Wait" until="'2008-12-24T18:00+01:00'"/>
```

Listing 28: Aus Beispielmodell generierte Wait Aktivität

3.6.7 Keine Operation

Die *Empty-Aktivität* ist eine Anweisung ohne Operation. Sie wird beispielsweise verwendet um einen Fehler abzufangen und dabei keine Aktion zur Fehlerbehandlung auszuführen.

[BPEL4WS, S. 58]. Dementsprechend einfach ist die Syntax, wie nachfolgend in Listing 29 dargestellt:

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

Listing 29: BPEL-Syntax der Empty Aktivität

Zur Modellierung ist ein *„Empty“*-Element vorhanden (Abb. 41), aus dem der nachfolgende BPEL-Code (Listing 30) generiert wird.

Abbildung 21: Elementtyp der Empty Aktivität

```
<empty name="Empty"/>
```

Listing 30: Aus Beispielmodell generierte Empty Aktivität

3.7 Strukturierte Aktivitäten

Als strukturierte Aktivitäten werden in BPEL Aktivitäten bezeichnet, die mehrere Aktivitäten beinhalten können und diese gliedern bzw. einen Kontrollfluss darauf definieren. Diese Aktivitäten sind *Sequence*, *Switch*, *While*, *Pick* und *Flow*. Dementsprechend gibt es im „ARIS SOA Designer“ für jede dieser Aktivitäten einen Elementtyp (Abbildung 22).

Abbildung 22: Elementtypen für strukturierte Aktivitäten

Die Aktivitäten, die sich innerhalb einer solchen Aktivität befinden, können atomare oder wiederum strukturierte Aktivitäten sein. Der Einfachheit wegen wird in den nachfolgenden Beispielen meist eine Empty Aktivität verwendet. Um Aktivitäten zu einer strukturierten Aktivität hinzuzufügen, werden diese entweder explizit mit der umfassenden Aktivität verbunden oder zur besseren Lesbarkeit „darin eingefügt“. Das bedeutet, dass die Verbindung nicht mehr explizit angezeigt wird, jedoch immer noch impliziert vorhanden ist. Dies wird anhand der folgenden Beispiele noch veranschaulicht.

3.7.1 Sequence Aktivität

Unter einer Sequenz versteht man eine Reihe von Aktivitäten, die nacheinander ablaufen. Die *Sequence Aktivität* beinhaltet eine Menge von Aktivitäten, welche in der gegebenen Reihenfolge ausgeführt werden. Sie endet mit der Beendigung der letzten Aktivität [BPEL4WS, S.59].

```
<sequence standard-attributes>
  standard-elements
  activity+
</sequence>
```

Listing 31: BPEL-Syntax der Sequence Aktivität

Zur Modellierung einer Sequenz mit dem „ARIS SOA Designer“ gibt es zwei Möglichkeiten. So können zwei Aktivitäten, die nacheinander ausgeführt werden sollen, direkt über eine Verbindung des Typs *„occurs before“* (bzw. je nach Leserichtung *„occurs after“)* verbunden werden. Diese ist durch einen entsprechenden Pfeil dargestellt ist. Die zweite Möglichkeit ist die Verwendung des *„Sequence“*-Elementes . Hierbei wird die erste Aktivität der zu modellierten Sequenz mit dem *„Sequence“*-Elementes verbunden. Der Typ dieser Beziehung ist *„starts with“*. Die folgenden Aktivitäten werden dann entsprechend ihrer Reihenfolge mit einer *„occurs before“*/*„occurs after“*-Verbindung in Beziehung zueinander gesetzt. Die Möglichkeit über das *„Sequence“*-Element dient hauptsächlich der Modellierung einer Sequence innerhalb einer anderen strukturierten Aktivität und um die Lesbarkeit bei verschachtelten Sequenzen zu erhöhen.

Die Abbildung 23 zeigt die Möglichkeiten der Modellierung einer Sequenz. Links ist ein Prozess dargestellt, bei dem alle „Aktivität“-Elemente mittels *„occurs before“*-Verbindung geordnet wurden. In der mittleren Form ist das *„Sequence“*-Element mit `'Activity1'` über *„starts with“* verbunden. Die Aktivitäten `'Activty1'` und `'Activity2'` sind mit einer *„occurs before“*-Beziehung verbunden. Die rechte Darstellung ist analog zur mittleren, jedoch wird die Verbindung zwischen den Aktivitäten und dem Sequence-Element nicht explizit über einen Pfeil angezeigt, sondern nur intern verwendet. Dazu muss das grafische *„Sequence“*-Symbol entsprechend vergrößert werden. Die darin enthaltenen Aktivitäten werde dann „hinein“ gesetzt. Die Startaktivität der Sequenz muss ebenfalls über *„starts with“* festgelegt werden. Diese Möglichkeit, der Vergrößerung des Symbols und „Einfügen“ der enthaltenen Aktivitäten ist für alle strukturierten Aktivitäten vorhanden.

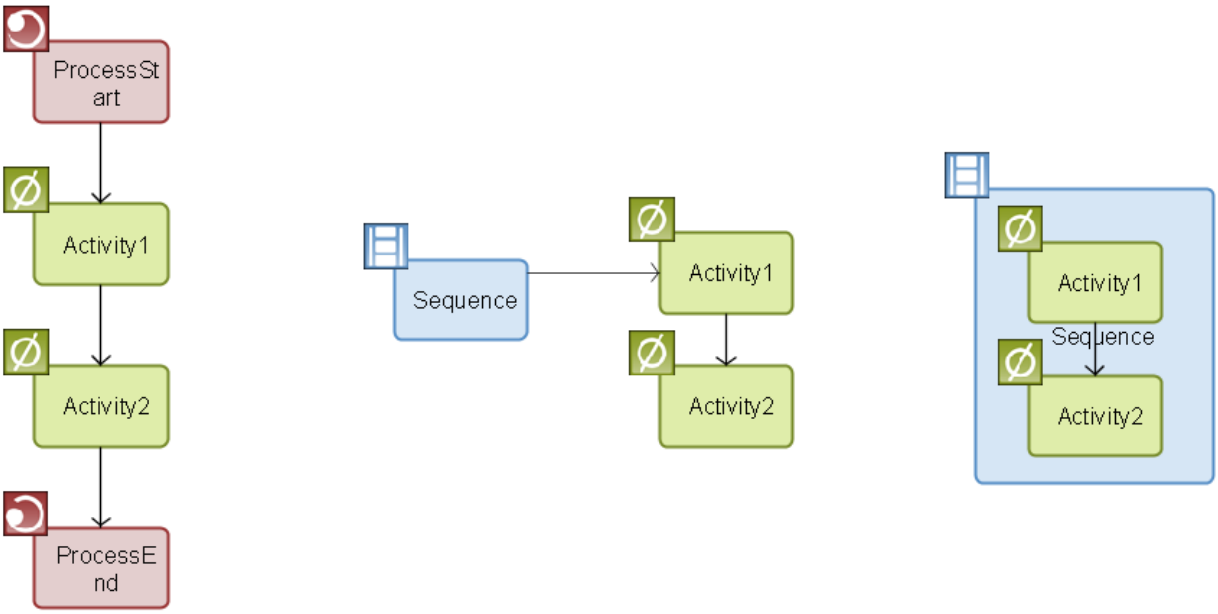

Abbildung 23: Modellierungsmöglichkeiten für Sequence

Der erzeugte BPEL-Code ist bei allen der drei Formen derselbe und wird in Listing 32 angegeben. Er verwendet die modellierten Aktivitäten in der korrekten Reihenfolge und entspricht der Spezifikation.

```
<sequence name="Sequence">
   <empty name="Activity1"/>
   <empty name="Activity2"/>
</sequence>
```

Listing 32: Aus Beispielmodell generierte Sequence Aktivität

3.7.2 Verzweigungen mit Fallunterscheidung

Mit der *Switch Aktivität* bietet BPEL die Möglichkeit innerhalb eines Prozess eine Verzweigung aufgrund boolescher Bedingungen zu verwenden. Sie beinhaltet einen oder mehrere Zweige, wovon der erste dessen Bedingung erfüllt ist, ausgewählt und ausgeführt wird. Zusätzlich gibt es noch einen optionalen *„Otherwise“*-Zweig, der genommen wird, wenn keine Bedingung der sonstigen Zweige erfüllt ist. [BPEL4WS, S.59]

```
<switch standard-attributes>
   standard-elements
   <case condition="bool-expr">+
      activity
   </case>
   <otherwise>?
      activity
   </otherwise>
</switch>
```

Listing 33: BPEL-Syntax der Switch Aktivität

Zur Modellierung einer Switch-Aktivität wird das entsprechende *„Switch“*-Element verwendet und mit den Aktivitäten verbunden, die jeweils in einem Zweig ausgeführt werden sollen. Der Typ dieser Verbindung ist dabei *„has Case“*. Über die BPEL-Attribute der Verbindung wird die Bedingung, wann dieser Zweig ausgeführt werden soll, festgelegt. Dazu dient das Attribut *„Condition Expression“*, womit die boolesche Bedingung für diesen Zweig angegeben wird. Wird das Attribut *„Default“* auf 'wahr' gesetzt, so spiegelt dies den optionalen Otherwise-Zweig wieder. Zusätzlich wird über das Attribut *„Sequence Number“* die Reihenfolge festgelegt, in der die einzelnen Bedingungen auftauchen und damit bei der Ausführung überprüft werden. Dazu wird ein nummerischer Wert eingegeben.

Die Abbildung 24 zeigt eine Switch-Aktivität mit drei Zweigen sowie die gewählten Attribute für den Zweig, der `Activity2` beinhaltet. Die Attribute für `Activity1` wurden entsprechend gesetzt (`sequence order` auf 1), für `Activity3` wurde 'Default' ausgewählt.

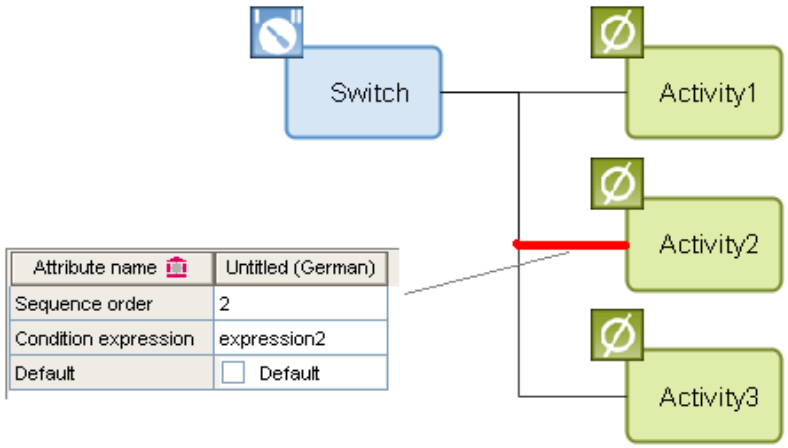

Abbildung 24: Switch-Aktivität

Der daraus erzeugte BPEL-Code ist nachfolgend dargestellt. Die modellierten Zweige mit ihren Bedingungen wurden korrekt umgesetzt.

```
<switch name="Switch">

  <case condition="expression1">
    <empty name="Activity1"/>
  </case>

  <case condition="expression2">
    <empty name="Activity2"/>
  </case>

  <otherwise>
    <empty name="Activity3"/>
```

```
  </otherwise>
</switch>
```

Listing 34: Aus Beispielmodell generierte Switch Aktivität

Eine weitere Möglichkeit der Modellierung ist einer Switch Aktivität ist in Abbildung 25 dargestellt. Sie unterscheidet sich jedoch nur in Darstellung. Sonst bleibt alles so, wie zuvor erläutert.

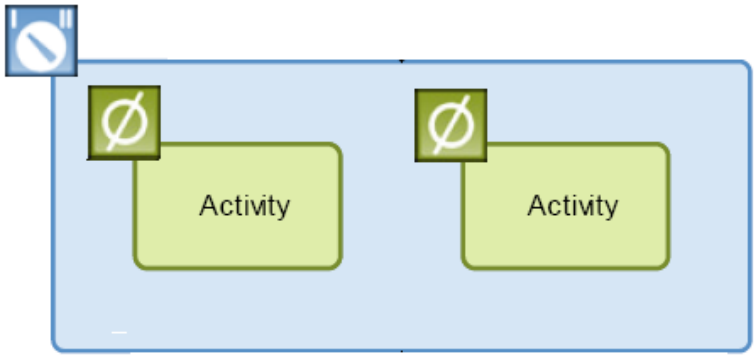

Abbildung 25: Switch-Modellierung

3.7.3 While

Die *While Aktivität* dient dazu, eine Aktivität solange wiederholt auszuführen bis eine boolesche Bedingung nicht mehr erfüllt wird [BPEL4WS, S.60]. Die Syntax ist in Listing 35 dargestellt. Sie beinhaltet also die auszuführende Aktivität und als Attribut noch die Bedingung, wie lange die Aktivität wiederholt werden soll.

```
<while condition="bool-expr" standard-attributes>
   standard-elements
   activity
</while>
```

Listing 35: BPEL-Syntax der While Aktivität

Zur Modellierung ein While Aktivität wird ein *„While“*-Element mit der Aktivität, die wiederholt werden soll, verbunden. Der Typ dieser Beziehung ist dabei *„performs“* (*while* `'performs'` *Aktivität*). Die Bedingung, wie lange die Aktivität ausgeführt werden soll, wird über das Attribut *„Path Condition“* des *„While“*-Elementes gesetzt.

Die Abbildung 26 zeigt eine einfache While Aktivität, die eine Empty Aktivität beinhaltet. Als Bedingung zur Ausführung wurde hierbei `'boolean-expression'` gesetzt. Zur Modellierung wurde die Verbindungen angezeugt. Es ist aber auch die Form möglich, dass die auszuführende Aktivität innerhalb des Symboles des While Aktivität, verwendet wird. (vgl. Abb. 25)

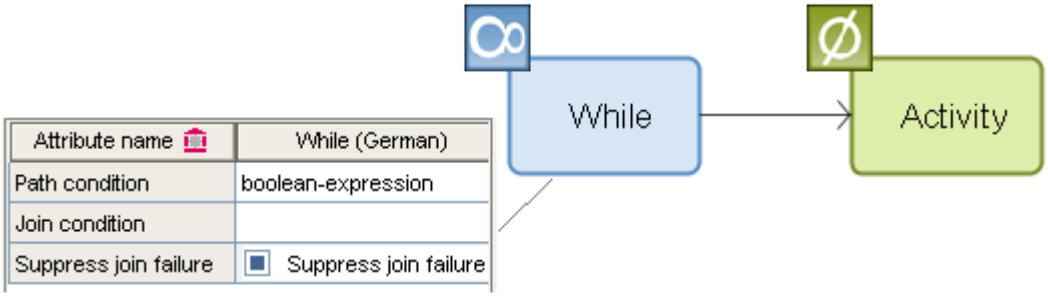

Abbildung 26: While-Aktivität

Daraus erzeugt der BPEL-Export folgenden spezifikationsgemäßen BPEL-Code, welcher der Modellierung entspricht.

```
<while condition="boolean-expression" name="While">
   <empty name="Activity"/>
</while>
```

Listing 36: Aus Beispielmodell generierte While Aktivität

3.7.4 Pick

Die *Pick Aktivität* beinhaltet mehrere Aktivitäten, die jeweils mit einem Ereignis verbunden sind. Sie wartet auf das Auftreten eines dieser Ereignisse und führt dann die entsprechende Aktivität aus. Die möglichen Ereignisse sind ankommende Nachrichten oder das Signal eines gesetzten Timers [BPEL4WS, S.61)]. Die Syntax ist im nachfolgenden Listing 46 angegeben.

```
<pick createInstance="yes|no"? Standard-attributes>
   standard-elements
   <onMessage partnerLink="ncname" portType="qname"
              operation="ncname" variable="ncname"?>+
      <correlations>?
         <correlation set="ncname" initiate="yes|no"?>+
      </correlations>
      activity
   </onMessage>
   <onAlarm (for="duration-expr" | until="deadline-expr")>*
      activity
   </onAlarm>
</pick>
```

Listing 37: BPEL-Syntax der Pick Aktivität

Um eine Pick-Aktivität zu modellieren, wird das *„Pick“*-Element mit den *„onMessage“*- bzw. *„onAlarm“*-Elementen verbunden. Diese sind jeweils mit einem den dazugehörigen *„Activity“*-Elementen verbunden ist.

Für die Modellierung des `<onMessage>`-Statements, werden zu dem entsprechenden *„On-*

Message"-Element noch zugehörige „*PartnerLink*"- und „*Operation*"-Objekte hinzugefügt. Diese sollten vorher definiert sein und geben an, über welchen Partner Link eine Nachricht erwartet wird. Zusätzlich können noch „*Variable*"- und „*CorrelationSet*"-Elemente mit dem „*OnMessage*"-Element in Beziehung gesetzt werden. Bei der Verwendung eines Correlation Sets muss das Attribut „*Initiate*" der Verbindung zwischen „*CorrelationSet*"- und „*onMessage*"-Element noch entsprechend gesetzt werden (siehe Abschnitt über Receive-Aktivität).

Zur Modellierung eines `<onAlarm>`-Statement, wird ein „*onAlarm*"-Element verwendet. Mittels dessen Attribute „*Type*" und „*Condition expression*" wird spezifiziert, wie lange gewartet werden soll. Das Attribut „*Type*" dient dabei zur Auswahl, ob es sich um einen `for` oder `until`-Timer handelt. Der Wert wird über das Attribut „*Condition expression*" festgelegt.

Die Abbildung 27 zeigt eine beispielhaft modellierte Pick Aktivität. Diese besitzt zwei Nachrichtenarten, die es verarbeiten kann, sowie einen Timer. Beim ersten

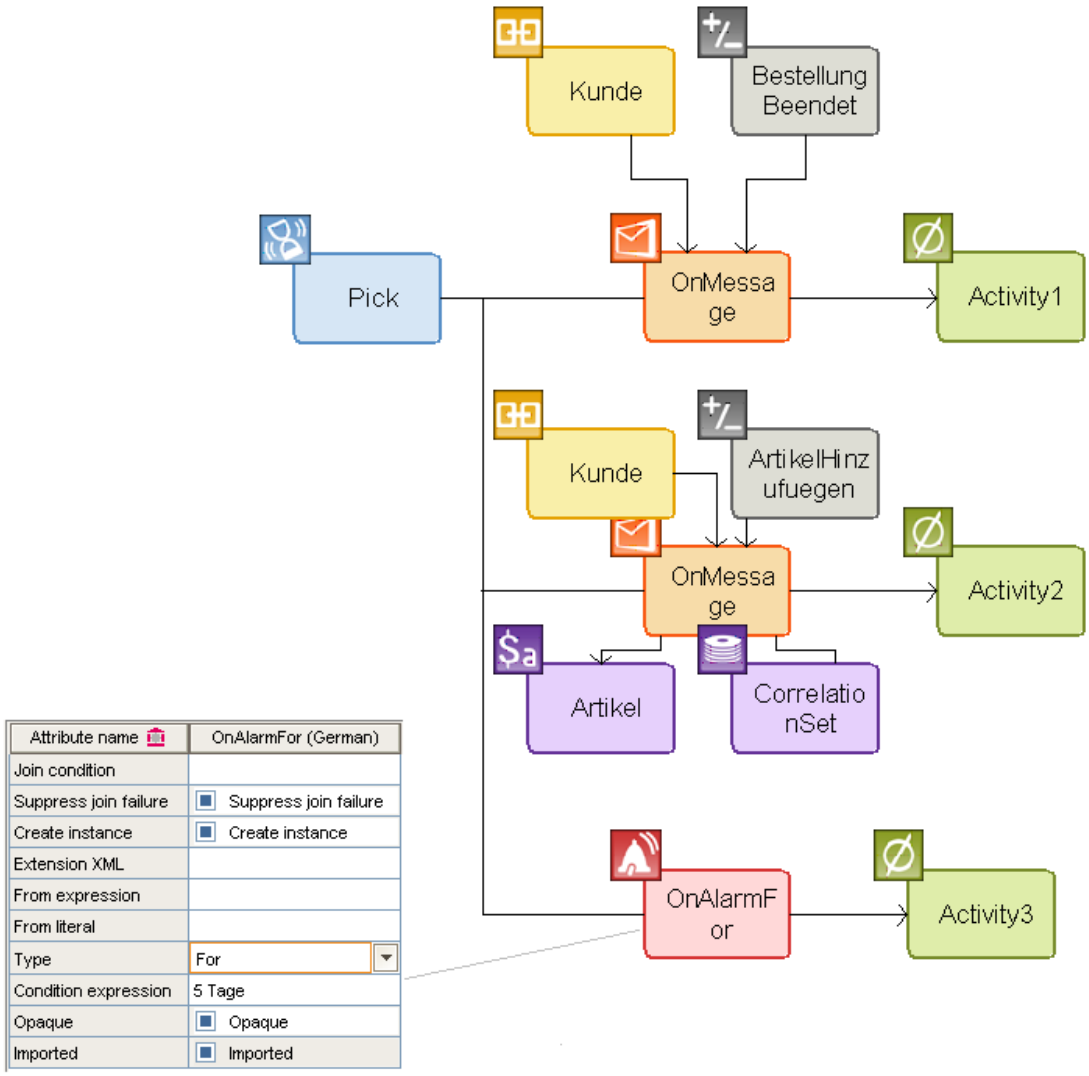

Abbildung 27: Beispielmodellierung einer Pick-Aktivität

„onMessage“-Objekt wurde der Partner Link `'Kunde'` und dessen Operation `'Bestellung-Beendet'` verwendet. Beim zweiten „onMessage“-Zweig, wird zudem noch eine Variable `'Artikel'` und ein Correlation Set `'CorrelationSet'` angegeben, wobei dessen Attribut *„Initiate“* auf 'no' gesetzt wurde (nicht abgebildet). Bei dem im Beispiel modellierten Timer wurde angegeben, dass er fünf Tage warten soll. Allerdings ist zu beachten, dass die *„Condition Expression“* hier nur der Einfachheit auf den syntaktisch falschen Ausdruck `'5 Tage'` gesetzt wurde.

Aus diesem modellierten Beispiel erzeugt die BPEL-Exportfunktion den in Listing 38 angegebenen Ausschnitt aus dem kompletten Prozess. Dieser entspricht dem Modell und erfüllt die Spezifikation.

```
<pick name="Pick">
   <onMessage operation="BestellungBeendet" partnerLink="Kunde"
              portType="tns:Bestellung">
      <empty name="Activity1"/>
   </onMessage>

   <onMessage operation="ArtikelHinzufuegen" partnerLink="Kunde"
              portType="tns:Bestellung" variable="Artikel">
      <correlations>
         <correlation set="CorrelationSet" initiate="no"‘/>
      </correlations>
      <empty name="Activity2"/>
   </onMessage>

   <onAlarm for="5 Tage">
      <empty name="Activity3"/>
   </onAlarm>
</pick>
```

Listing 38: Aus Beispielmodell generierte Pick Aktivität

3.7.5 Flow

Die *Flow Aktivität* (siehe Listing 39) dient der parallelen Ausführung von Aktivitäten und der Synchronisation dieser über so genannte Links [BPEL4WS, S.62]. Sie beinhaltet verschiedene Aktivitäten, welche nebenläufig ausgeführt werden. Um diese zu synchronisieren und in Abhängigkeit zueinander zu setzen, dienen die Standardelement `<source>` und `<target>` einer Aktivität.

```
<flow standard-attributes>
   standard-elements
   <links>?
      <link name="ncname">+
   </links>
   activity+
</flow>
```

Listing 39: BPEL-Syntax der Flow Aktivität

Mit dem „ARIS SOA Designer“ wird dies durch das *„Flow“*-Elementes modelliert. Diese kann die Aktivitäten beinhalten, welche innerhalb des Flows ausgeführt werden sollen. Um einen Link zwischen den *„Aktivität“*-Elementen zu realisieren, setzt man eine entsprechende Verbindung des Typs *„links“*, welche durch einen gestrichelten Pfeil angezeigt wird. Über das Attribut *„Transition Condition“* einer *„links“*-Verbindung kann entsprechendes Attribute des zugehörigen `<source>`-Elementes gesetzt werden.

Das Beispiel in Abbildung 28 zeigt ein Beispiel für einen Flow. Dieser beinhaltet die drei Aktivitäten `ActivityA`, `ActivityB` und `ActivyC`. Diese Aktivitäten `ActivityA` und `ActivityB` können sofort mit dem Start der Flow-Aktivität begonnen werden und nebenläufig ablaufen. Die Aktivität `ActivityC` ist über einen Link mit `ActivityA` verbunden. Die Modellierung im rechten Teil der Abbildung entspricht dabei genau, des Modells im linken

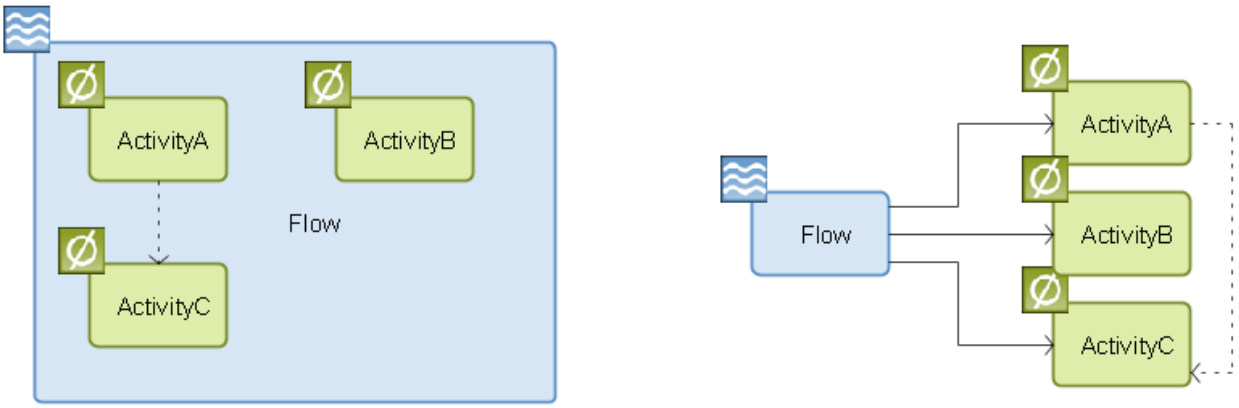

Abbildung 28: Flow-Aktivität

Teil. Jedoch wird links die Verbindung zwischen dem *„Flow“*-Element und den Aktivitäten nicht explizit angezeigt.

Aus diesen modellierten Flows generiert die BPEL-Exportfunktion jeweils denselben BPEL-Code, welcher in Listing 40 angezeigt wird. Darin ist der modellierte Link zwischen `'ActivityA'` und `'ActivityB'` wird korrekt als Link mit dem Namen `'ActivityAToActivityB'` erzeugt. Dieser Link wird in den entsprechenden `<empty>`-Elementen durch deren Unterelement `<source>` bzw. `<target>` richtig verwendet. Aus dem modellierten Prozesselement wurde also BPEL-Code erzeugt, welcher die Spezifikation erfüllt.

```
<flow name="Flow">
   <links>
      <link name="ActivityAToActivityC"/>
   </links>

   <empty name="ActivityA">
      <source linkName="ActivityAToActivityC"/>
   </empty>
   <empty name="ActivityB"/>
   <empty name="ActivityC">
      <target linkName="ActivityAToActivityC"/>
```

```
    </empty>
</flow>
```

Listing 40: Aus Beispielmodell generierte Flow Aktivität

3.8 Scopes

Ein Scope ist ein Unterblock eines Prozesses, der den darin stattfindenden Aktivitäten lokale Variablen, Correlation Sets, sowie Handler für Fehler-, Ereignis- und Kompensationsbehandlung festlegen kann [BPEL4WS, S. 69ff.]. Der grundsätzliche Aufbau ist ähnlich zu der eines Prozesses. Lediglich die fehlende Definitionsmöglichkeit von Partner Links und Partner unterscheiden die Prozess- und die Scope-Definition

```
<scope variableAccessSerializable="yes|no" standard-attributes>
  standard-elements
  <variables>?
    ...
  </variables>
  <correlationSets>?
    ...
  </correlationSets>
  <faultHandlers>?
    ...
  </faultHandlers>
  <compensationHandler>?
    ...
  </compensationHandler>
  <eventHandlers>?
    ...
  </eventHandlers>
activity
</scope>
```

Der ARIS SOA Designer verwendet zur Modellierung von Scopes die Elementtypen *„ScopeStart“* und *„ScopeEnd“*.

Abbildung 29: Elementtypen zur Definition von Scopes

Die Modellierung damit ist analog zu den *„ProcessStart“*- und *„ProcessEnd“*-Elementen. Zwischen den jeweiligen Start- und Endelement befindet sich der Prozessablauf, dieses Unterblockes. Die Definition der Variablen, Correlation Sets sowie der Fehler-, Kompensations- und Ereignisbehandlung innerhalb des Blockes wird durch Verbinden der entsprechenden Elemente mit dem „ScopeStart“-Element erreicht. Das Attribut `variableAccessSerializable` kann durch gleichnamiges Attribut des *„ScopeStart“*-Elementes gesetzt

werden.

3.9 Fehler und Kompensationsbehandlung

Zur Fehler- und Kompensationsbehandlung können in BPEL entsprechende FaultHandler bzw. CompensationHandler definiert werden. Die Modellierung wurde teilweise schon in vorangegangen Beispielen aufgezeigt (vgl. Abbildung 4 bzw. Abbildung 21 mit zugehörigem Listing 18). In diesem Abschnitt werden sie dennoch noch einmal kurz erläutert.

3.9.1 Kompensationsbehandlung

Ein CompensationHandler beinhaltet eine Aktivität, welche zur Kompensation eines Fehlers innerhalb eines Blockes ausgeführt wird. Dementsprechend enthält die Syntax nur diese Aktivität.

```
<compensationHandler>?
   activity
</compensationHandler>
```

Listing 41: BPEL-Syntax zur CompensationHandler Definition

Zur Modellierung eines CompensationHandler wird die kompensierende Aktivität mit dem Block, auf den sich der Handler bezieht verbunden. Also wird eine „Activity“-Element mit einem „ProcessStart“ (wie in Abb. 30) oder „ScopeStart“-Element verbunden. Der Typ dieser Verbindung ist dafür *„defines Compensation“*

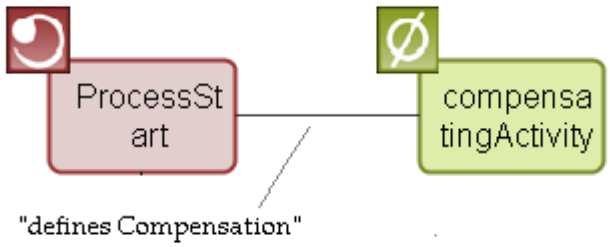

Abbildung 30: Modellierung eines CompensationHandlers

Dieses Modell erzeugt innerhalb eines des Prozesses folgenden BPEL-Code (Listing 42). Bei entsprechender Modellierung wird derselbe Ausschnitt auch innerhalb eines Scopes generiert. Dies entspricht also der Spezifikation.

```
<process...>
   <compensationHandler>
      <empty name="compensatingActivity"/>
   </compensationHandler>
</process>
```

Listing 42: Aus Beispielmodell generierter CompensationHandler

Der Aufruf eines CompensationHandlers erfolgt durch die *„Compensate“*-Aktivität. Diese

kann prinzipiell wie eine normale Aktivität verwendet werden. Sie gibt denen Namen eines Scopes an, dessen CompensationHandler aufgerufen werden soll.

```
<compensate scope="name"? Standard-attributes>
   standard-elements
</compensate>
```

Listing 43: BPEL-Syntax der Compensate Aktivität

Die Modellierung im „ARIS SOA Designer“ erfolgt analog dazu. Hierbei wird das *„Compensate“*-Element verwendet, welche an der gewünschten Stelle als Aktivität in den Prozess eingefügt wird. Dieses Element wird noch mit dem *„Scope“*-Start-Element verbunden, dessen CompensationHandler aufgerufen werden soll

Abbildung 31: Verwendung der Compensate Aktivität

3.9.2 Fehlerbehandlung

Zur Fehlerbehandlung kann innerhalb eines Blockes ein FaultHandler definiert werden (Listing 44). Dieser beinhaltet Aktivitäten, die bei dem Auftreten eines Fehler, ausgeführt werden sollen. Dazu wird zu einem Fehlernamen mit optionaler Fehlervariable, welche Information zu dem Fehler enthält, ein entsprechende Aktivität festgelegt. Über das optionale `<catchAll>`-Statement können sonstige nicht näher definierte Fehler aufgefangen werden.

```
<faultHandlers>?
<!-- there must be at least one fault handler or default -->
  <catch faultName="qname"? faultVariable="ncname"?>*
    activity
  </catch>
  <catchAll>?
   activity
  </catchAll>
</faultHandlers>
```

Listing 44: BPEL-Syntax zur FaultHandler Definition

Zur Modellierung eines solchen FaultHandlers mit dem ARIS SOA Designer steht das *„Catch“*-Element zur Verfügung. Dieses wird mit einem „ProcessStart“ oder *„ScopeStart“*-Element verbunden um den FaultHandler dort zu definieren. Die im Fehlerfall auszuführende Aktivität wird ebenfalls mit dem *„Catch“*-Element verbunden. Um festzulegen, welcher Fehler aufgefangen wird und um diesen näher zu beschreiben, können auch

noch entsprechende „Fault“- und „Variable“-Elemente mit dem *„Catch“*-Element verbunden werden.

Um innerhalb eines Blockes ein `<catchAll>` zu modellieren, wird die darin auszuführende Aktivität mit einem „ProcessStart“ oder „ScopeStart“-Element verbunden. Der Typ dieser Verbindung ist dabei *„catches All“*.

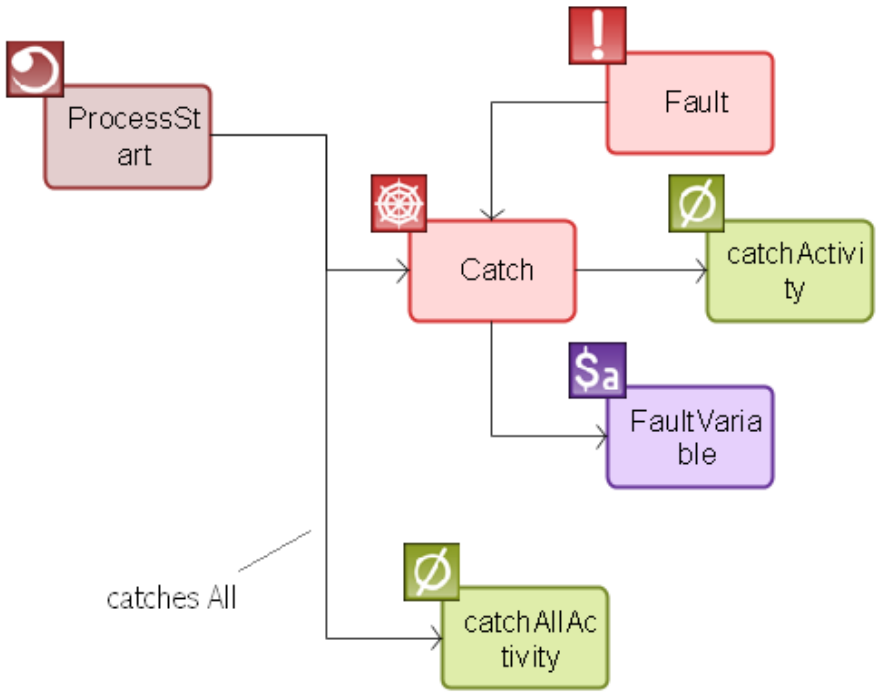

Abbildung 32: Modellierung eines FaultHandlers

3.9.3 Ereignisbehandlung

Die Ereignisbehandlung gibt dem Prozess oder einem Unterblock die Möglichkeit auf bestimmte Ereignisse zu reagieren. Ein Event Hander besteht aus verschiedenen `<onMessage>` und `<onAlarm>`-Elementen. Diese beschreiben auf welche Nachrichten, bzw. auf welchen Timer mit welcher Aktivität reagiert wird.

```
<eventHandlers>?
<!-- there must be at least one onMessage or onAlarm handler -->
   <onMessage partnerLink="ncname" portType="qname" operation="ncname"
              variable="ncname"?>*
      <correlations>?
         <correlation set="ncname" initiate="yes|no">+
      </correlations>
      activity
   </onMessage>
   <onAlarm for="duration-expr"? until="deadline-expr"?>*
      activity
   </onAlarm>
</eventHandlers>
```

Listing 45: BPEL-Syntax der Event Handler Definition

Zur Modellierung eines EventHandlers mit dem „ARIS SOA Designer“, genügt es die entsprechenden *„onMessage“* bzw. *„onAlarm“*-Elemente entweder mit dem *„ProcessStart“*- oder eines *„Scope“*-Elements zu verbinden. Die jeweiligen Ereignisse müssen dann noch genauer, spezifiziert werden, also beispielsweise Partner Link und Correlation angegeben, bzw. der Timer gesetzt werden. Dies geschieht, wie im Abschnitt über die Pick Aktivität dargestellt und wird deshalb hier nicht weiter vertieft.

4 Fazit

Die Modellierung eines BPEL-Prozesses mit dem „ARIS SOA Designer“ erweist sich meistens als recht intuitiv und entspricht den Erwartungen. Für so gut wie jedes BPEL-Bestandteil, ist in dem Werkzeug ein entsprechendes Elementsymbol vorhanden, mit dem die Modellierung erfolgt. Zur Modellierung von BPEL-Konstrukten, bei welchen bestimmte BPEL-Elemente verwendet werden, die Unterelemente oder Attribute eines anderen sind, genügt es häufig die entsprechenden Symbole miteinander zu verbinden. Dadurch ist die Modellierung sehr nahe an der Sprache BPEL, wodurch sie für Personen mit BPEL-Erfahrung recht einfach zu erlernen sein dürfte. Lediglich einige wenige Attribute sind etwas kompliziert zu finden, da sie teils den Elementen und teils einer Verbindung zugeordnet werden. Zudem besitzen manche Attribute im „ARIS SOA Designer“ einen anderen Namen als in BPEL, wodurch die Modellierung etwas erschwert wird.

Es werden sämtliche Konstrukte der Sprache BPEL umgesetzt. Der aus den Modellen erzeugte BPEL-Code entspricht in der Regel der Spezifikation. Die hier beschriebenen Fehler bei der Definition von Partner Links treten scheinbar in neueren Versionen nicht mehr auf.

Allerdings lässt sich durch eine unkorrekte Modellierung auch fehlerhafter BPEL-Code generieren, wie dies in einigen Fällen angedeutet wurde. Der aus dem Pozessmodell generierte BPEL-Prozess entspricht dabei zwar dem Modell, jedoch nicht der Spezifikation. Hier wäre es sinnvoll, beim Export nach BPEL zumindest noch entsprechende Warnungen auszugeben. Ein weiterer Schritt wäre sogar, das Anbieten einer Möglichkeit zur Validierung von BPEL-Prozessmodelle vor dem Export. Dies leistet das ARIS Werkzeug zum Beispiel bei der Modellierung ereignisgesteuerter Prozessketten, welche vor einem Export in BPEL-Prozessmodelle überprüft werden können.

Der Hauptnachteil ist aber, dass das ARIS Toolset bisher nur den veralteten BPEL Standard der Version 1.1 unterstützt und noch keine BPEL 2.0 Implementierung enthält.

Abschließend lässt sich aber sagen, dass der „ARIS SOA Designer“ ein gutes Werkzeug zur Erstellung von BPEL-Prozessen ist. Als Möglichkeit zur reinen Prozessmodellierung auf fachlicher Ebene ist BPEL aufgrund seiner Technikorientierung nicht sonderlich geeignet, und auch gar nicht dafür entwickelt. Deshalb dürfte die Erstellung eines BPEL-Prozesses, die nur durch eine entsprechende Modellierung erfolgt, in der Praxis sehr selten vorkommen. Vielmehr werden die Prozesse zunächst mittels anderer Notationen erstellt, welche dann in BPEL überführt werden. Genau hier setzt auch das ARIS Werkzeug an. Es erlaubt die fachliche Modellierung von Prozessen und die Überführung in BPEL-Prozessmodelle. Für solche Modelle ist die, im Rahmen dieser Arbeit vorgestellte Modellier von BPEL-Prozessen gut einsetzbar.

Die BPEL-Modellierungsmöglichkeit der ARIS Platform bietet den IT-Verantwortlichen innerhalb eines Unternehmens eine relativ einfache grafische Modellierungsmöglichkeit zur Erweiterung von Prozessmodellen. Dies können bereits von fachlichen Verantwortlichen auf anderer Ebene erstellt wurden. Damit bietet sich also eine Möglichkeit, aus einer Prozessmodellierung auf eher abstrakter Ebene, in relativ einfacher Art und Weise ausführbare BPEL-Prozesse zu erzeugen.

Literaturverzeichnis

[ARISDS70] IDS Scheer AG, "*ARIS SOA Designer 7.0*", Manual Version 1.0, Oktober 2006.

[ARISPROD] IDS Scheer AG, "*ARIS Platform - Produktbroschüre*", 2008. Verfügbar unter: http://www.ids-scheer.de/set/6473/Produktbroschuere%202008-07.pdf

[ARISWP] IDS Scheer AG, "*ARIS Platform*", System White Paper, Mai 2008. Verfügbar unter: http://www.ids-scheer.de/set/6473/ARIS_Platform_SWP_de_2008-05.pdf

[BKV05] G. Bischoff, R. Kersten, T. Vetter. "*Vergleich von BPEL-Workflow Modllierungstools*". Fachstudie, Universität Stuttgart, 2005

[BPEL2.0] OASIS Group. "*Web Services Business Process Execution Language Version 2.0*", April 2007. Verfügbar unter: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[BPEL4WS] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weerawarana. "*Business Process Execution Language for Web Services - Version 1.1*", Mai 2003. Verfügbar unter: http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

[BPMN] Object Management Group. "*Business Process Modelling Notation*", Februar 2006. Verfügbar unter: http://www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%201-0%20Spec%2006-02-01.pdf

[IDS] Webauftritt der Firma IDS Scheer AG, http://www.ids-scheer.de/

[Klü07] J. Klückmann, "*In 10 Schritten zur Business-Driven SOA*", ARIS Expert Paper, Januar 2007. Verfügbar unter: http://www.ids-scheer.de/set/6473/ARIS_Expert_Paper-SOA-10_Schritte_zur_SOA_Klueckmann_2007-01_de.pdf

[KNS92] G. Keller, M. Nüttgens, A.-W. Scheer. " *Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)“*", in: Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi), Universität des Saarlandes, Heft 89, Januar1992

[Kop05] O. Kopp. "*Abbildung von EPKs nach BPEL anhand des Prozessmodellierungswerkzeugs Nautilus*". Diplomarbeit, Universität Stuttgart, 2005

[Sch07] D. Schumm. "*A Graphical Tool for Modeling BPEL 2.0 processes*". Studienarbeit, Universität Stuttgart, 2007

[Sch07b] O. Schmelzle. "*Transformation von annotierten Geschäftsprozessen nach BPEL*". Masterarbeit, Gottfried Wilhelm Leibniz Universität Hannover, Mai 2007

[Sch08] D. Schumm. "*Graphische Modellierung von BPEL Prozessen unter Verwendung der BPMN Notation*". Diplomarbeit, Universität Stuttgart, 2008

[Sch97] A. W. Scheer. "*Wirtschaftsinformatik: Referenzmodelle für industrielle Geschäftsprozesse*". Springer, 7. Auflage, 1997

[Sta06] J. Staud. "*Geschäftsprozessanalyse: Ereignisgesteuerte Prozessketten und objektorientierte Geschäftsprozessmodellierung für Betriebswirtschaftliche Standardsoftware*". Springer, 3. Auflage, 2006

[WSDL] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana. "*Web Services Description Language (WSDL) 1.1*", März 2001. Verfügbar unter: http://www.w3.org/TR/2001/NOTE-wsdl-20010315

Alle angegebenen Links wurden zuletzt am 3. August 2008 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit
selbstständig verfasst und nur die
angegebenen Quellen verwendet zu haben.

(Stefan Varnhorn)