Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Konzept und Entwicklung einer BPEL Compliance Testsuite in BPELUnit

Bachelorarbeit

im Studiengang Informatik

von

Stephan Kiesling

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Prof. Dr. Ing. Gabriele von Voigt
Betreuer: Leif Singer

Hannover, 20.08.2009

Zusammenfassung

Die Ausführung von BPEL-Prozessen auf BPEL-Engines geschieht, je nach Engine, auf unterschiedliche Art und Weise. Dies liegt daran, dass der BPEL-Standard nicht von jeder Engine vollständig implementiert und von vielen anders interpretiert wird.

Deshalb beschäftigt sich diese Bachelorarbeit mit der Erstellung einer Compliance Testsuite, um die Einhaltung des BPEL-Standards von den Engines messbar zu machen. Dabei ist das Ziel der Compliance Testsuite das Sicherstellen der Interoperabilität der BPEL-Prozesse.

Es wird zuerst ein Verfahren angegeben, das die Erstellung der Testfälle beschreibt, welche anschließend aus verschiedenen Quellen abgeleitet werden. Diese Testfälle werden zu einer Compliance Testsuite zusammengefasst.

Ist die Testsuite fertig gestellt, wird eine BPEL-Engine durch sie getestet. Das Testergebnis wird anschließend erläutert.

Danksagung

An dieser Stelle möchte ich allen danken, die mich bei der Erstellung dieser Bachelorarbeit unterstützt haben. Insbesondere möchte ich meinem Betreuer Leif Singer danken, der mir die gesamte Zeit, rund um die Uhr mit Rat und Tat zur Seite gestanden hat.

Inhaltsverzeichnis

1. Einleitung **1**
- 1.1. Motivation . . . 1
- 1.2. Problemstellung . . . 1
- 1.3. Struktur der Arbeit . . . 2

2. Grundlagen **3**
- 2.1. SOA . . . 3
  - 2.1.1. SOA-Grundlagen . . . 3
  - 2.1.2. Services . . . 3
- 2.2. BPEL (Business Process Execution Language) . . . 4
- 2.3. Software-Tests . . . 7
  - 2.3.1. Unit-Tests . . . 7
  - 2.3.2. Compliance Testsuites . . . 8
  - 2.3.3. BPELUnit . . . 8

3. Entwicklung eines Verfahrens zur Ableitung von Testfällen **9**
- 3.1. Vorgehensweisen und Prinzipien zur Ableitung von Testfällen . . . 9
  - 3.1.1. Überblick . . . 9
  - 3.1.2. Äquivalenzklassen bilden . . . 9
  - 3.1.3. Benutzerumfrage über BPEL-Engines . . . 11
  - 3.1.4. Glassboxverfahren . . . 12
  - 3.1.5. Effizienz- und Effektivitätsforderungen . . . 13
- 3.2. Besondere Anforderungen an eine Compliance Testsuite . . . 14
- 3.3. Verfahren zur Ableitung von Testfällen . . . 16
  - 3.3.1. Einleitung . . . 16
  - 3.3.2. Probleme . . . 16
  - 3.3.3. Angabe des Verfahrens . . . 18

4. Priorisierung und Ableitung der Testfälle **20**
- 4.1. Priorisierung in zwei Stufen . . . 20
- 4.2. Priorisierung vor Ableitung der Testfälle . . . 21
- 4.3. Ableitung der Testfälle . . . 22
- 4.4. Priorisierung nach Ableitung der Testfälle . . . 37
- 4.5. Erstellung der BPELUnit-Tests . . . 38
  - 4.5.1. Distanzierung von BPELUnit . . . 38
  - 4.5.2. Lösung mit Apache Ant . . . 38

5. Evaluation der Compliance Testsuite **40**
- 5.1. Testdurchführung . . . 40

5.2. Probleme bei der Wahl einer zweiten BPEL-Engine 40
5.3. Apache ODE BPEL-Engine im Test 41
5.4. jBPM BPEL-Engine im Test 42

6. Zusammenfassung **43**
6.1. Kritische Würdigung 43
6.2. Ausblick 44
6.3. Fazit 45

A. Testfälle **47**

B. Standardfehler **57**

C. Abkürzungen für Testfälle **58**

D. Übersicht der Apache ODE Testergebnisse **59**

E. Compact Disk **60**

1. Einleitung

1.1. Motivation

Die Business Process Execution Language spielt im Rahmen der serviceorientierten Architektur und speziell in der Entwicklung sogenannter Webservices eine große Rolle. Sie dient dabei zur Orchestrierung dieser Webservices, welche von BPEL-Engines kompiliert und ausgeführt werden.

Es gibt eine Vielzahl verschiedener BPEL-Engines unterschiedlicher Anbieter. Alle implementieren und interpretieren den BPEL-Standard auf ihre eigene Art und Weise. Dies erschwert den Austausch und Vergleich einzelner, ausführbarer Geschäftsprozesse, die mit unterschiedlichen BPEL-Engines erstellt wurden.

Wünschenswert wäre eine einheitliche Implementierung des BPEL-Standards aller BPEL-Engines, wodurch die Portabilität der BPEL-Prozesse gewährleistet wäre.

1.2. Problemstellung

Ziel dieser Bachelorarbeit ist es, eine Compliance Testsuite zu entwickeln, die diese Portabilität der BPEL-Prozesse messbar macht. Konkret soll sie in der Lage sein, für eine beliebige BPEL-Engine zu testen, in wie weit diese dem BPEL-Standard entspricht.

Dahinter steckt die folgende Idee: Entsprechen zwei BPEL-Engines dem Standard zu vollem oder hohem Ausmaß, so wird davon ausgegangen, dass sie zueinander kompatibel sind.

Um dies zu erreichen, soll die Compliance Testsuite eine Ansammlung von Tests enthalten, die automatisiert ausgeführt werden können. Diese Tests sollen eine BPEL-Engine auf die geforderten Eigenschaften hin überprüfen. Dabei soll die Testsuite auf Basis der Ingenieursprinzipien erstellt werden.

Es soll dementsprechend ein Verfahren angegeben werden, mit dessen Hilfe Testfälle abgeleitet und erstellt werden können. Die Testfälle sollen priorisiert und anschließend gemäß der Priorisierung implementiert werden.

Zum Abschluss sollen zwei BPEL-Engines durch die Compliance Testsuite getestet und bezüglich des Testergebnisses miteinander verglichen werden. Die Apache ODE BPEL-Engine soll eine der beiden Engines sein.

1.3. Struktur der Arbeit

In dieser Arbeit werden im Anschluss an dieses Kapitel zuerst grundlegende Begriffe erklärt, die das Verständnis der Methoden und Verfahren dieser Arbeit fördern sollen.

Danach wird in Kapitel 3 ein Verfahren zur Ableitung von Testfällen hergeleitet. Dabei werden Methoden angegeben, mit deren Hilfe Testfälle erstellt werden können, und besondere Anforderungen aufgezeigt, die an eine Compliance Testsuite gestellt werden. Vor diesem Hintergrund wird anschließend ein konkretes Verfahren zur Ableitung von Testfällen angegeben.

In Kapitel 4 wird dieses Verfahren angewendet, um Testfälle abzuleiten, welche in tabellarischer Form aufgelistet werden. Zudem wird erklärt, wie die Priorisierung der Testfälle aussieht und wie sich die Testfälle zu einer Compliance Testsuite zusammenfügen.

Das darauffolgende Kapitel 5 dokumentiert die ausgeführten Tests der BPEL-Engines durch die Compliance Testsuite und vergleicht die Testergebnisse miteinander.

Im letzten Kapitel wird rückblickend auf Probleme hingewiesen, die während der Erstellung dieser Arbeit enstanden sind. Es werden Möglichkeiten in Aussicht gestellt, die sich durch diese Arbeit auftun und ein Fazit angegeben, welches sich abschließend noch einmal mit der Thematik der Arbeit beschäftigt.

2. Grundlagen

2.1. SOA

2.1.1. SOA-Grundlagen

Eine serviceorientierte Architektur ist eine Softwarearchitektur, die zur Strukturierung von Diensten (engl. Services) einzelner Unternehmen gedacht ist. Diese Dienste sind meist über das gesamte Unternehmen verteilt und der zentrale Bestandteil der Architektur.

Vorgesehen wird eine Menge von lose gekoppelten Services, die meist entlang von Geschäftsprozessen zusammengestellt sind.

Das Ziel einer serviceorientierten Architektur ist das Senken der Kosten auf lange Sicht. Die Idee dabei ist alle vorhandenen Services zu sammeln, wodurch nach wiederholter Entwicklung einer Anwendung alle Services bereits vorhanden sind und lediglich korrekt angeordnet werden müssen.

Weitere Ziele sind das Erreichen einer hohen Flexibilität und Wartbarkeit bezüglich Änderungen der Geschäftsprozesse. Dadurch sollen sich Anwendungen an geänderte Anforderungen schneller und leichter anpassen lassen.

Die Mittels einer serviceorientierten Architektur zusammengestellten Dienste können beispielsweise im Internet angeboten und von anderen Benutzern oder Unternehmen verwendet werden. Die Kommunikation zwischen so angebotenen Diensten erfolgt über Protokolle wie zum Beispiel SOAP.

Ein Reiseunternehmen kann auf die Art in der Lage sein, ein Hotelzimmer mittels eines vom Hotel angebotenen Services automatisch zu buchen.

2.1.2. Services

Definition Service, nach Lübke[1]: „Ein Service ist eine im Netzwerk verfügbare, lose gekoppelte Softwarekomponente, welche wohl-definierte Schnittstellen implementiert und mit standardisierten Protokollen aufgerufen werden kann.“

Definition Webservice, nach Lübke[1]: „Ein Webservice ist ein Service, der von anderen Softwaresystemen über ein Netzwerk und möglicherweise sogar dem Internet über SOAP aufrufbar ist.“

2.2. BPEL (Business Process Execution Language)

Die Business Process Execution Language, kurz BPEL, dient zur Beschreibung von ausführbaren Geschäftsprozessen. Die Aktivitäten der Geschäftsprozesse werden dabei durch Webservices dargestellt. Die Sprache basiert auf XML und ist aus der Web Services Flow Language von IBM, sowie XLANG von Microsoft entstanden.

Das Ziel von BPEL ist nicht die Programmierung einer Anwendung im herkömmlichen Sinne. Vielmehr geht es um die Einordnung einzelner Komponenten in einen Arbeitsablauf. Dieser Arbeitsablauf wird von dem zugrunde liegenden Geschäftsprozess bestimmt. Die Einordnung der Komponenten wird auch Orchestrierung genannt, wobei die Komponenten die Webservices sind. Sie können zusammenarbeiten, Nachrichten austauschen, sowie andere Webservices aufrufen.

Die BPEL-Prozesse kommunizieren ausschließlich mit Webservices. Eine Interaktion mit Menschen oder anderen Anwendungen wird nicht unterstützt und findet nur über die Webservices statt, die als Schnittstelle dienen können. Konzepte, die versuchen die Interaktion mit Menschen einzubeziehen, werden beispielsweise mit BPEL4People umgesetzt. Die Erweiterung BPELJ erlaubt die Einbindung von Java-Code in die einzelnen Webservices.

BPEL-Prozesse selbst besitzen sogenannte Aktivitäten, die maßgeblich für die Ausführung der Prozesse sind. Diese Aktivitäten sind aufgeteilt in Basisaktivitäten und strukturierte Aktivitäten. Die Verschachtelung der Aktivitäten geschieht rekursiv. Es gibt also nur eine Aktivität auf der obersten Ebene.

Basisaktivitäten sind atomare Aktivitäten, die die Basis von BPEL bilden. Sie führen bestimmte Aufgaben aus, ohne dabei aus anderen Aktivitäten zu bestehen.

Zu den Basisaktivitäten zählen:

Assign verändert den Wert einer Variablen.

Compensate ist eine Aktivität, die aufgerufen werden kann, wenn ein Fehler aufgetreten ist. In ihr sollten Aktivitäten aufgerufen werden, die die Auswirkungen der fehlerhaften Aktivität rückgängig machen.

CompensateScope funktioniert wie die *compensate*-Aktivität. Es kann zusätzlich ein Ziel angegeben werden.

Empty ist eine leere Aktivität.

Exit beendet unmittelbar alle Aktivitäten.

Invoke ruft einen anderen Webservice synchron oder asynchron auf.

Receive wartet auf das Empfangen einer Nachricht eines anderen Webservices.

Reply schickt eine Nachricht an einen wartenden Webservice.

Rethrow wird in einem Fault Handler benutzt und wirft einen aufgetretenen Fehler weiter.

Throw wirft explizit einen selbst definierten Fehler.

Validate bestätigt den Wert einer Variablen in Bezug auf die entsprechende XML- oder WSDL-Definition.

Wait legt eine Zeitspanne oder einen Zeitpunkt fest. Es wird dann gewartet, bis die Zeitspanne abgelaufen oder der Zeitpunkt erreicht ist.

Strukturierte Aktivitäten beinhalten andere Aktivitäten und steuern so den Kontrollfluss. Mit ihrer Hilfe ist es möglich, Aktivitäten rekursiv aufzubauen.

Zu den strukturierten Aktivitäten zählen:

Flow ermöglicht eine graphbasierte sowie eine parallele Abarbeitung von Aktivitäten. Die enthaltenen Aktivitäten starten gleichzeitig, sobald die Flow-Aktivität ausgeführt wird. Sie endet, wenn alle enthaltenen Aktivitäten beendet wurden. Die Aktivitäten in einer *flow*-Aktivität werden durch Links miteinander verknüpft.

ForEach besitzt einen Start- und einen Endwert. Die beinhaltete Aktivität wird entsprechend der positiven Differenz der beiden Werte ausgeführt.

If überprüft eine bestimmte Bedingung und entscheidet dann, welche Aktivität ausgeführt werden soll.

Pick funktioniert ähnlich wie die *if*-Aktivität. Anstelle von Bedingungen werden allerdings Ereignisse abgefragt. Ereignisse sind hierbei das Empfangen einer Nachricht oder das Erreichen einer Zeitspanne.

RepeatUntil ruft eine Aktivität solange auf, bis eine bestimmte Bedingung erfüllt ist. Die Bedingung wird nach dem Aufruf überprüft. Die beinhaltete Aktivität wird folglich mindestens einmal aufgerufen.

Scope bildet eine in sich geschlossene logische Einteilung. Er kann eigene Variablen, eigene *PartnerLink*-Elemente, sowie eigene *compensationHandler*- und *faultHandler*-Elemente beinhalten.

Sequence erlaubt das sequentielle Abarbeiten einer oder mehrerer Aktivitäten.

While ruft eine Aktivität kontinuierlich auf, solange eine bestimmte Bedingung erfüllt ist. Die Bedingung wird vor dem Aufruf überprüft.

Dabei ist die *invoke*-Aktivität eine der wichtigsten, da sie dazu dient, andere Webservices aufzurufen, was der zentrale Aspekt der Webservices ist.

Neben den eben angegebenen Aktivitäten kann ein BPEL-Prozess noch aus anderen Elementen bestehen. Diese Elemente sollen an dieser Stelle kurz erklärt werden, um einen groben Überblick zu geben:

Extensions dienen zur Einbindung von Erweiterungen in einen BPEL-Prozess. Als Erweiterung kann zum Beispiel ein neues Attribut oder eine erweiterte Aktivität bezeichnet werden.

Imports dienen zum Importieren von XML- und WSDL-Schemata.

Partner Links modellieren die BPEL-Prozesse, mit denen der zugehörige BPEL-Prozess interagieren möchte.

Message Exchanges definieren einen Nachrichtenaustausch zwischen zwei BPEL-Prozessen. Die Angabe von *messageExchange*-Elementen ist optional und dient lediglich zur Verdeutlichung eines Nachrichtenaustausches.

Variables dienen zur Deklaration von Variablen.

Correlation Sets dienen zur genauen Adressierung einer bestimmten Instanz eines BPEL-Prozesses. Wird ein BPEL-Prozess neu aufgerufen, so wird eine Instanz des BPEL-Prozesses erzeugt, mit der gearbeitet wird. Um diese Instanz eindeutig zu benennen, werden *correlationSet*-Elemente benötigt.

Fault Handlers dienen zur Fehlerbehandlung und werden aufgerufen, wenn ein Fehler aufgetreten ist.

Event Handlers werden aufgerufen, wenn bestimmte Ereignisse eingetreten sind. Ein Ereignis kann dabei entweder eine eingehende Nachricht oder ein zeitlich bedingter Alarm sein.

Jedes Unternehmen besitzt seine eigene Art, Geschäftsprozesse mit BPEL umzusetzen. Damit die Entwicklung der BPEL-Prozesse einheitlich verläuft, benötigt man eine Standardisierung. Im April 2003 wurde die Organization for the Advancement of Structured Information Standards (OASIS) damit beauftragt, einen Standard für BPEL zu entwickeln, der BPEL4WS 1.1 genannt wurde. Im April 2007 wurde schließlich der aktuelle Standard WS-BPEL 2.0 festgelegt.

Die Ausführung der Prozesse von BPEL geschieht durch die sogenannten BPEL-Engines. Das Einbringen der Prozesse in die Engines wird „Deployen“ genannt. Abhängig von der verwendeten BPEL-Engine werden unterschiedliche Deployment-Informationen bereitgestellt. Einige Engines sind aufgelistet:

Apache ODE ist eine Open Source BPEL-Engine, die sowohl den BPEL-Standard 1.1 als auch 2.0 unterstützt und Apache Tomcat verwendet. ODE steht für Orchestration Director Engine.

ActiveVOS ist die kommerzielle Weiterentwicklung von ActiveBPEL. Mit ActiveVOS können serviceorientierte Anwendungen komplett erstellt werden. Dies geht von der Modellierung über die Orchestrierung bis hin zum Deployment. Es unterstützt die Standards BPMN, WS-BPEL 2.0, sowie BPEL4People. Zudem gibt es die Möglichkeit, die BPMN-Modelle direkt in BPEL-Code umzuwandeln.

Websphere Process Server dient zum Deployen und Ausführen von Prozessen, die Services orchestrieren. WPS wird von IBM verkauft und speziell für serviceorientierte Architekturen angeboten.

Oracle BPEL Server bietet zusätzlich zu seiner Funktion als BPEL-Engine ein grafisches Tool, das in Eclipse oder JDeveloper eingebunden werden kann, um Services zu orchestrieren. Der Oracle BPEL Server wird in der Oracle SOA Suite mitgeliefert, die ebenfalls kommerziell vertrieben wird.

jBPM ist eine kostenlose BPEL-Engine, die den JBoss Application Server benötigt und den BPEL-Standard 2.0 unterstützt.

2.3. Software-Tests

2.3.1. Unit-Tests

Ein Unit-Test oder auch Modultest ist ein wesentlicher Bestandteil in der Softwareentwicklung und dient zur Qualitätssicherung. Er testet im Rahmen des Testprozesses die einzelnen Komponenten bzw. Module, isoliert von äußeren Einflüssen, auf Fehler. Speziell bei objektorientierter Programmierung sind damit meist einzelne Klassen oder Methoden gemeint.

Durch einen Unit-Test wird die jeweilige Komponente ausgeführt. Die Rückgabewerte, welche die einzelnen Methoden der Komponente oder die Komponente selbst liefern, werden mit den erwarteten Ergebnissen verglichen. Auf diese Weise kann festgestellt werden, ob die Komponente korrekt arbeitet. Das Problem besteht darin, alle oder möglichst viele Methoden mit unterschiedlichen Parametern aufzurufen. Dabei ist es wichtig, die Parameter geschickt zu wählen und keine mögliche Situation zu vergessen.

Es gibt zwei wesentliche Prinzipien, die Tests durchzuführen:

Blackbox-Tests überprüfen allgemein, ob eine bestimmte Komponente die ihr auferlegten Anforderungen erfüllt. Dabei sollte jede Anforderung durch mindestens einen Testfall abgedeckt werden und jeder Testfall möglichst viele Anforderungen abdecken. Die Testfälle sollten zudem ein besonderes Augenmerk auf Randbedingungen und falsche Parameter legen. Derart ausgelegte Tests nennt man Negativ-Tests. Zudem sollten sie nicht vom selben Programmierer angelegt werden, um Objektivität zu gewährleisten. Sollten die gegebenen Anforderungen nicht ausreichend, vollständig oder genau genug sein, so ist der erfolgreiche Durchlauf eines Blackbox-Tests kein Indiz dafür, dass die Komponente korrekt arbeitet. Der Programm-Code braucht bei dieser Variante nicht bekannt zu sein (Blackbox).

Glassbox-Tests beachten die Struktur und speziell den Programm-Code der Komponente. Dadurch können Schwächen im Code lokalisiert und gezielt durch Tests geprüft werden. Voraussetzung hierbei ist, dass der Programm-Code bekannt ist (Glassbox). Insbesondere besteht die Möglichkeit, einen Kontrollflussgraphen der Komponente anzufertigen, mit dessen Hilfe man möglichst alle strukturellen Variationen durch Tests abdecken will. Man spricht hierbei von Überdeckung oder auch Coverage. Mittels des Graphen kann man die Anweisungs-, Zweig-, Bedingungs- oder Pfadüberdeckung ermitteln, welche von den Testfällen erreicht wird.

Unit-Tests können vor, während oder nach dem Erstellen einer Komponente erstellt und sowohl manuell als auch automatisch ausgeführt werden. Wird ein Unit-Test vor oder parallel zur Erstellung einer Komponente angefertigt, so spricht man von einer testgesteuerten Programmierung oder Test First Development.

Nachdem die Unit-Tests durchgeführt wurden, werden die Integrationstests ausge-

führt, welche die jeweiligen Module auf ihre Zusammenarbeit untereinander prüfen. Die Ausführung der Integrationstests geschieht meist manuell.

2.3.2. Compliance Testsuites

Unter einer Compliance Testsuite versteht man eine Ansammlung von Tests und gegebenenfalls auch Tools und Dokumentationen. Mit diesen Tools und Tests kann überprüft werden, ob eine bestimmte Anwendung oder ein bestimmtes Objekt bestimmte Standards einhält oder einer bestimmten Spezifikation entspricht. Insbesondere kann so überprüft werden, inwieweit sich ein Objekt oder eine Anwendung für eine vorgesehene Arbeit eignet.

2.3.3. BPELUnit

BPELUnit ist ein Framework zum automatisierten Testen von BPEL-Prozessen, welches zur xUnit Familie gehört. Das bekannteste Mitglied dieser Familie ist JUnit für Java. BPELUnit funktioniert prinzipiell ähnlich.

Der Testablauf besteht aus Deployment, Test und Undeployment und ist unabhängig von der verwendeten BPEL-Engine. Der Test kann gelingen oder fehlschlagen. Schlägt er fehl, so unterscheidet BPELUnit zwischen Error und Failure, wobei Failures erwartet werden und Errors unerwartet auftreten.

Zudem kann BPELUnit die Testabdeckung berechnen und Mocks für verwendete Webservices erstellen.

BPELUnit sendet Nachrichten an einen BPEL-Prozess und überprüft anschließend, ob die zurückgesendeten Nachrichten den erwarteten Werten entsprechen. Die Nachrichten die ein BPEL-Prozess und BPELUnit austauschen, liegen in XML-Format vor. Für die Überprüfung der Nachrichten wird dabei XPath verwendet.

XPath ist eine Abfragesprache mit deren Hilfe Teile eines XML-Dokuments selektiert werden können und steht für XML Path Language. Die XPath Version 1.0 wurde im Januar 2007 von Version 2.0 abgelöst.

BPELUnit selektiert also mittels XPath bestimmte Teile der empfangenen Nachricht und ist so in der Lage, diese Teile mit den erwarteten Werten zu vergleichen.

3. Entwicklung eines Verfahrens zur Ableitung von Testfällen

3.1. Vorgehensweisen und Prinzipien zur Ableitung von Testfällen

3.1.1. Überblick

Im Zuge des fortschreitenden Prozesses der Softwareentwicklung entstand das Bedürfnis, Software zu testen. Dabei wurde festgestellt, dass es bei einer umfangreicheren Software so gut wie nie möglich ist, sie komplett zu testen.

Daher wurden Methoden entwickelt, mit deren Hilfe Testfälle erstellt werden, die möglichst viele Eigenschaften der Software möglichst effizient und schnell testen. Mitlerweile gibt es sehr viele Verfahren diese Testfälle zu erstellen.

Aus dieser großen Menge von Verfahren sind natürlich nicht alle dazu geeignet, Testfälle für eine Compliance Testsuite abzuleiten. Hinzu kommt noch, dass die Verfahren bereits im Hinblick auf die Realisierung mit BPELUnit gewählt werden müssen, da einige Verfahren nicht mit BPELUnit realisiert werden können. Auf der anderen Seite gibt es wiederum Verfahren, die sich speziell auf die Entwicklung von BPEL-Prozessen und das Testen mit BPELUnit beziehen. In diesem Abschnitt sollen die relevanten Verfahren vorgestellt und erläutert werden.

3.1.2. Äquivalenzklassen bilden

Das Äquivalenzklassenverfahren hat seine Wurzeln in der Mathematik. Wie der Name schon sagt liegt ihm die Bildung von Äquivalenzklassen zugrunde, die mittels einer Äquivalenzrelation beschrieben werden.

Auf mathematischer Ebene werden Objekte mit sich ähnelnden Eigenschaften betrachtet. Eine Äquivalenzrelation ist eine Relation, die gewisse Mindestanforderungen erfüllt um eine Gleichwertigkeit von Objekten zu beschreiben. Eine Äquivalenzrelation erfüllt folgende Bedingungen:

Reflexivität: $a \sim a$

Jedes Objekt ist zu sich selbst äquivalent.

Symmetrie: $a \sim b \Leftrightarrow b \sim a$

Ist a äquivalent zu b, so ist auch b äquivalent zu a und umgekehrt.

Transitivität: $a \sim b$ und $b \sim c \Rightarrow a \sim c$

Ist a äquivalent zu b und b äquivalent zu c, so ist auch a äquivalent zu c.

Als Äquivalenzklasse eines Objektes a bezeichnet man nun die Menge aller Objekte, die zu a äquivalent sind. Die einzelnen Objekte einer Äquivalenzklasse werden als Repräsentanten bezeichnet.

Dieses Prinzip soll anhand eines Beispiels verdeutlicht werden:

Wir betrachten die Menge aller wissenschaftlichen Mitarbeiter an einer Universität. Als Äquivalenzklassen kann man beispielsweise die Fakultäten wählen. Ein Mitarbeiter ist somit ein Repräsentant der Äquivalenzklasse beziehungsweise der Fakultät. Zwei Mitarbeiter sind äquivalent zueinander, wenn sie an der selben Fakultät arbeiten.

In der Informatik wird dieses Prinzip für die Erstellung von Softwaretests angewendet. Als Grundmenge wird die Menge aller möglichen Eingabewerte für einen Parameter definiert. Dabei wird als Mindestanforderung der Äquivalenzrelation die Gleichheit der Fehler gefordert, die nach Ausführung der Software mit entsprechenden Objekten entstehen. Ob zwei Objekte die gleichen Fehler verursachen, lässt sich in der Regel nicht oder nur sehr schwer beweisen, wodurch ein geringes Risiko besteht, Fehler zu übersehen.

Bezüglich der so definierten Äquivalenzrelation sind zwei Objekte äquivalent, wenn sie im Softwaretest die gleichen Fehler erzeugen. Als Objekt wird hierbei ein zulässiger oder unzulässiger Eingabewert bezeichnet. Somit bestehen die Äquivalenzklassen aus möglichen Eingabewerten eines Parameters, die bezüglich der oben genannten Relation äquivalent sind

Besitzt eine Software mehrere Parameter, so müssen für jeden Parameter Äquivalenzklassen erstellt werden. Die Äquivalenzrelation bleibt allerdings immer gleich.

Anschließend wird für jede so ermittelte Äquivalenzklasse mindestens ein Testfall erstellt, was man als Minimalforderung bezeichnet. Der Testfall erwartet dabei einen beliebigen Repräsentanten der Äquivalenzklasse. Es sollte auch darauf geachtet werden, dass jede Äquivalenzklasse nur ein einziges Mal getestet wird. Dies wird als Effizienzprinzip bezeichnet.

Zum Verständnis wird an dieser Stelle ein kurzes Beispiel angegeben:

Betrachtet wird ein einfacher Kaffeeautomat, der nur über eine einzige Kaffeesorte verfügt. Der Automat nimmt Münzen bis einschließlich 2 Euro entgegen. Es wird so viel Kaffee ausgeschenkt, wie bezahlt wurde, minimal aber 50 ml und maximal 400 ml. 2 ml Kaffee kosten 1 Cent. Der Automat kann kein Geld zurückgeben.

Die Software des Automaten besitzt also einen einzigen Parameter, nämlich die Summe des eingeworfenen Geldes. Bezüglich dieses Parameters und bezüglich der oben genannten Äquivalenzrelation ergeben sich nun die folgenden Äquivalenzklassen:

ÄQUIVALENZKLASSE	ERKLÄRUNG
1 Cent - 24 Cent	Es wurde zu wenig bezahlt und kein Kaffee ausgegeben.
25 Cent - 2 Euro	Es wurde genug bezahlt. Der Automat schenkt Kaffee aus.
ab 2 Euro	Es wurde zu viel bezahlt. Das restliche Geld verfällt.

Für die drei Äquivalenzklassen könnte man nun die Eingabewerte 20 Cent, 1 Euro und 3 Euro wählen. Die Äquivalenzklassen sind korrekt gewählt, weil für jeden Eingabewert aus den jeweiligen Klassen vermutlich der gleiche Fehler auftritt.

Speziell in Bezug auf die Äquivalenzklassenmethode bietet sich das Grenzwertverfahren an. Es wird davon ausgegangen, dass die Eingaben Gültigkeitsbereiche besitzen. Es werden Testfälle gezielt auf die Grenzen dieser Bereiche abgestimmt, da bei diesem Verfahren davon ausgegangen wird, dass Fehler besonders häufig an genau diesen Grenzen auftreten.

In Bezug auf das Äquivalenzklassenverfahren werden insbesondere die Grenzen der jeweiligen Äquivalenzklassen betrachtet. Bei einem nicht trivialen Programm existieren derartige Grenzen auf Grund der Forderung der Fehlergleichheit für die Äquivalenzklassen. Ein triviales Programm ist hierbei ein Programm, dass ausschließlich oder aber nie einen Fehler ausgibt.

Für einen Grenzwerttest würden im obigen Beispiel folglich die Eingabewerte 1 Cent, 24 Cent, 25 Cent, 200 Cent, sowie 201 Cent gewählt werden. Zudem wird oft auch der Eingabewert 0 Cent getestet, obwohl dieser im praktischen Gebrauch nicht vorkommen sollte. Mit diesem Test möchte man lediglich überprüfen, ob der Automat gegen Systemfehler so gut wie möglich geschützt ist.

Das Äquivalenzklassenverfahren ist ein bewährtes Verfahren, wenn es um die Gewinnung von Testfällen geht. Es verhindert die mehrfache und somit überflüssige Erstellung des gleichen Testfalls. Dadurch kann die Menge aller Testfälle etwas zusammenschrumpfen, was Zeit bei der Priorisierung und Arbeit für die Erstellung des entsprechenden BPELUnit-Tests spart. Aus diesen Gründen eignet sich das Äquivalenzklassenverfahren gleich doppelt für die noch folgende Ableitung der Testfälle.

Das Grenzwertverfahren eignet sich nur bedingt für die Ableitung der Testfälle. Es ist nur einsetzbar, wenn die Äquivalenzklassen klare Grenzen und mehr als ein Element besitzen. Dies ist bei dieser Art von Testfällen nicht immer gegeben. Sollten die Grenzen allerdings klar ersichtlich sein und die Äquivalenzklassen mehr als ein Element enthalten, so ist das Grenzwertverfahren ein Verfahren, das zusätzliche Sicherheit bietet und eignet sich deshalb ebenso wie das Äquivalenzklassenverfahren.

3.1.3. Benutzerumfrage über BPEL-Engines

Wird eine Software bereits benutzt, ist nicht garantiert, dass sie fehlerfrei läuft. In vielen Fällen treten während des Betriebs Defekte auf. Zudem sind manchmal in der aktuellen Version der Software Funktionen noch nicht implementiert.

Dies gilt auch für die zu testenden BPEL-Engines. Da sie von unterschiedlichen Herstellern entwickelt wurden, ist anzunehmen, dass sie auch unterschiedliche Defekte

aufweisen und an unterschiedlichen Stellen Probleme verursachen. Genau diese Fehler und Probleme soll die Compliance Testsuite zum Vorschein bringen.

Da BPEL-Engines bereits veröffentlicht sind, gibt es schon Benutzer, die diese Engines verwenden und Erfahrungen mit Problemen und Fehlern gemacht haben. Dazu zählen nicht nur Probleme, die durch das Verwenden einer einzelnen Engine entstanden sind, sondern auch Kompatibilitätsprobleme zwischen einzelnen Engines.

Diese Kompatibilitätsprobleme können beispielsweise entstehen, wenn unterschiedliche Benutzer ihre BPEL-Prozesse austauschen und anschließend mit ihrer eigenen BPEL-Engine ausführen wollen.

Durch eine Umfrage unter diesen Benutzern sollen die angesprochenen Erfahrungen über Fehler und Probleme gesammelt werden. Existieren derartige Erfahrungen, so liegt es nahe, Testfälle aus ihnen zu konstruieren, um so gezielt auf bereits existierende Schwachstellen zu testen. Mit anderen Worten testet man auf Fehler, von denen man bereits weiß, dass sie existieren.

Erfahrungen über Probleme und Fehler beziehen sich allerdings nicht nur auf die BPEL-Engine, in der sie aufgetreten sind. Vielmehr können sie auf problematische Eigenschaften des BPEL-Standards hinweisen, die schwierig umzusetzen sind. Sie decken also potentielle Schwachstellen der BPEL-Engines im Allgemeinen auf, wodurch sie umso wertvoller für die Compliance Testsuite werden.

3.1.4. Glassboxverfahren

Das Glassboxverfahren ist, wie in Unterabschnitt 2.3.1 beschrieben, eine Methode der Softwaretests, bei dem die Struktur des Prüfgegenstands bekannt sein muss.

Dies ist bei eigener, selbst erstellter Software immer gegeben. Da allerdings Testfälle für die Compliance Testsuite hergeleitet werden sollen, ist klar, dass die betreffende Software nicht die Compliance Testsuite selbst, sondern viel mehr die BPEL-Engine ist.

Um das Glassboxverfahren anzuwenden, muss also der Quellcode der BPEL-Engines bekannt sein. Da der Quellcode nicht von allen BPEL-Engines bekannt ist, kann man dieses Verfahren nur auf die quelloffenen BPEL-Engines anwenden.

Glassboxtests werden in den meisten Fällen dazu benutzt, die Struktur eines Programms zu überprüfen. Das bedeutet, dass diese Struktur auf bestimmte Kriterien überpüft wird. Dabei wird die Anweisungs-, Zweig-, Bedingungs- oder Pfadüberdeckung gemessen. Die Messung bezieht sich auf die erstellten Testfälle. Jeder Testfall kann mehrere Anweisungen, Zweige, Bedingungen oder Pfade überdecken. Die am Ende gemessene Überdeckung hängt somit von den Testfällen ab.

Diese Vorgehensweise eignet sich nicht besonders gut um Testfälle für die Compliance Testsuite herzuleiten, da kein besonderer Wert auf eine möglichst hohe Überdeckung gelegt wird. Außerdem soll die BPEL-Engine selbst nicht auf ihre Korrektheit getestet werden, sondern auf ihre Kompatibilität zum BPEL-Standard.

Daher wird die Aufmerksamkeit auf das konkrete Finden von Schwachstellen gelegt. Dazu gehört zum Beispiel das Benutzen von Fehler-Datenbanken.

Diese Idee lässt sich ebenfalls gut auf die Spezifikation des BPEL-Standards beziehen. Dort ist es besonders gut möglich, anfällige Stellen zu identifizieren und konkret daraufhin zu testen. Anfällige Stellen können zum Beispiel undeutlich oder mehrdeutig formulierte Äußerungen sein.

Dabei bleibt die Frage offen, inwiefern sich mögliche Unterschiede in den BPEL-Engines bezüglich solcher Stellen auf die Korrektheit der BPEL-Engine auswirken können. Schließlich ist es nicht Aufgabe der BPEL-Engine, Bestandteile des BPEL-Standards zu interpretieren oder sogar zu erraten. Letzten Endes ist dies aber dennoch ein Punkt, der sich auf die Kompatibilität einzelner BPEL-Engines zueinander auswirkt und sollte daher getestet werden.

Es wird an dieser Stelle nicht möglich sein, den kompletten Quellcode der BPEL-Engines zu durchsuchen. Der Zeitumfang der Bachelorarbeit ist begrenzt, wodurch es sinnvoller ist, die Zeit an anderer Stelle einzusetzen, beispielsweise bei der Umsetzung möglichst vieler BPELUnit-Tests.

3.1.5. Effizienz- und Effektivitätsforderungen

Natürlich sollte die Priorität dieses Verfahrens zur Ableitung von Testfällen darauf liegen, korrekte Testfälle zu erstellen, was anders umschrieben bedeutet, Testfälle zu erstellen, die tatsächlich den Anforderungen in den Punkten Relevanz und Richtigkeit entsprechen.

Ein Beispiel für nicht vorhandene Relevanz wäre ein Testfall, der ausschließlich Anforderungen aus der Spezifikation des BPEL-Standards überprüft, die optional sind. Wie genau deutlich wird, ob Anforderungen optional oder überhaupt relevant sind, wird im nachfolgenden Abschnitt genauer erläutert.

Ein Beispiel für einen inkorrekten Testfall wäre ein solcher, der Eigenschaften fordert, welche im Standard nicht auftauchen oder gefordert werden.

Allerdings ist ein weiterer wichtiger Punkt die Effizienz der Testfälle. Für die Anforderungen, die an eine Compliance Testsuite gestellt werden, ist er zwangsläufig von keiner großen Bedeutung. Die hier zu entwickelnde Compliance Testsuite soll „lediglich“ eine BPEL-Engine auf ihre Kompatibilität zum Standard hin überprüfen. In welcher Zeit sie das tut oder wieviele und welche Ressourcen sie dabei benötigt, ist kein Bestandteil dieser Anforderung.

Für den Benutzer selbst, der die Compliance Testsuite bedienen will, ist dieser Aspekt allerdings sehr wohl von Bedeutung. Er muss wissen, ob das System, auf dem die Compliance Testsuite läuft, diese auch ausführen kann. Zudem sollte die Ausführung nicht zu lange dauern. Auf Grund dieser Faktoren ist es lohnenswert, sich mit dem Thema Effizienz auseinander zu setzen.

Tut man dies, so entsteht allerdings das folgende Problem: Durch die Kombination

einzelner Testfälle zu einem großen Testfall, könnten sich diese Testfälle negativ beeinflussen. Dies muss nicht zwangsweise geschehen, ist aber ein Risikofaktor.

Der Testfall an sich beschreibt lediglich das Verhalten eines Prüfgegenstandes unter den angegebenen Voraussetzungen und Umständen und gibt ein erwartetes Ergebnis vor. Werden nun zwei oder sogar mehrere Testfälle zu einem einzigen Testfall zusammengeführt, so ändern sich auch die Voraussetzungen, die die einzelnen Testfälle gefordert haben.

Es wäre beispielsweise denkbar, dass ein Abschnitt aus einem bereits ausgeführten Testfall einen Systemfehler verursacht hat. Dieser Systemfehler könnte alle anschließenden Testfälle beeinträchtigen und im schlimmsten Fall deren Ergebnisse verfälschen.

Es gilt also zwischen Sicherheit und somit Korrektheit der Testfälle, sowie ihrer Effizienz abzuwiegen. Beides zusammen ist nicht vollständig zu erreichen.

Der erste intuitive Gedanke könnte nun sein, einen Kompromiss zu finden, mit anderen Worten einen Mittelweg zu finden. Das würde für die Testfälle bedeuten, eine Zusammenlegung zu einem gewissen Grad der Verschachtelung zuzulassen. Somit würde man ein gewisses Risiko eingehen.

Führt man diesen Gedankengang weiter, wird jedoch folgendes klar: Wenn die Testfälle nicht das leisten, was sie leisten sollen oder zu leisten vorgeben, ist es auch nicht von Bedeutung, wie effizient sie sind. Daher sollte dem Prinzip der Effizienz immer das der Korrektheit vorgezogen werden.

Aus diesem Grund ist es zwangsläufig verpflichtend, in erster Linie auf die Korrektheit der Testfälle zu achten und erst an zweiter Stelle die Effizienz der Testfälle zu berücksichtigen. Die beste Lösung wäre also, jeden Testfall getrennt zu betrachten, um maximale Sicherheit im Sinne der Compliance Testsuite zu gewährleisten.

3.2. Besondere Anforderungen an eine Compliance Testsuite

Das Ziel dieser Arbeit ist die Entwicklung einer Compliance Testsuite, mit deren Hilfe überprüft werden kann, zu welchem Grad eine BPEL-Engine den BPEL-Standard implementiert.

Die Testsuite selbst besteht aus einer Sammlung von Tests, die mittels BPELUnit erstellt werden und einen möglichst hohen Abdeckungsgrad des BPEL-Standards erreichen sollen.

Natürlich können solche Tests nicht willkürlich und ohne Schema aufgestellt und geschrieben werden. Es bedarf einem oder mehreren Mustern, um dies systematisch zu erreichen.

Um die BPELUnit-Tests systematisch herzuleiten und zu erstellen, werden zunächst allgemeine Testfälle erstellt, die von BPELUnit unabhängig sein sollen und sich aus

diversen Faktoren zusammensetzen. Dies hat den Vorteil, dass die erstellten Testfälle, sofern sie korrekt erstellt wurden und der an sie gestellten Aufgabe entsprechen, auch auf andere Weise implementiert oder genutzt werden können. Wie die Erstellung der Testfälle konkret geschieht, wird im nachfolgenden Abschnitt 3.3 verdeutlicht.

In diesem Abschnitt soll nun zuerst erklärt werden, welche besonderen Anforderungen an eine Compliance Testsuite gestellt werden. Daraus ergibt sich implizit, welche Eigenschaften die BPELUnit-Tests und somit auch die Testfälle erfüllen müssen, da sie, wie oben beschrieben, miteinander verknüpft sind. Die Compliance Testsuite arbeitet genau dann zuverlässig und korrekt, wenn auch die Testfälle und BPELUnit-Tests zuverlässig und korrekt arbeiten. Umgekehrt gilt somit: Wenn die BPELUnit-Tests oder die Testfälle nicht korrekt sind, kann auch die Compliance Testsuite nicht korrekt arbeiten.

Zur Verdeutlichung liegt der Aussage die folgende Äquivalenz zugrunde: Compliance Testsuite arbeitet korrekt $\Leftrightarrow$ Testfälle sind korrekt $\Leftrightarrow$ BPELUnit-Tests sind korrekt und arbeiten korrekt

Inzwischen ist vielleicht klar geworden, dass sich die Testfälle nicht direkt aus dem Standard ergeben, sondern daraus abgeleitet werden müssen. Wie dies genau geschieht, wird im folgenden Abschnitt genauer erläutert. Um die Vorgehensweise und das Verfahren besser zu verstehen, wird deshalb an dieser Stelle zuerst erklärt, wie der Standard WS-BPEL 2.0 generell definiert und aufgebaut ist.

Der BPEL-Standard definiert in erster Linie die XML-basierte Syntax, die ein BPEL-Prozess haben soll. Dabei definiert er sowohl die generelle Struktur eines Geschäftsprozesses als auch die Struktur einzelner Aktivitäten. Darüber hinaus werden im BPEL-Standard Eigenschaften festgelegt, die eine bestimmte Aktivität besitzen muss.

Eine BPEL-Engine, die einen BPEL-Prozess verarbeiten will, muss in der Lage sein, die Syntax des Prozesses zu interpretieren. Wenn eine BPEL-Engine dem BPEL-Standard gerecht werden will, so muss sie jeden BPEL-Prozess, der ebenfalls dem BPEL-Standard entspricht, interpretieren können. Sie muss also sowohl die Syntax des Prozesses akzeptieren als auch die Semantik.

In der Spezifikation des BPEL-Standards werden Schlüsselbegriffe definiert, die ein hohes Maß an Aussagekraft über die Eigenschaften besitzen, die im BPEL-Standard angegeben werden. Damit keine Unklarheiten oder Verwirrungen entstehen, wurden diese Begriffe in einem RFC (Request for Comments) festgehalten.[2]

Die Schlüsselbegriffe lauten: „MUST“, „REQUIRED“, „SHALL“, „MUST NOT“, „SHALL NOT“, „SHOULD“, „RECOMMENDED“, „SHOULD NOT“, „NOT RECOMMENDED“, „MAY“ und „OPTIONAL“.

Diese Begriffe nehmen eine Gewichtung der Definitionen vor, zu denen sie gehören, was speziell für die Compliance Testsuite von besonderer Bedeutung ist. Wie die Begriffe zu lesen sind und was das für die Compliance Testsuite bedeutet, soll an dieser Stelle kurz erklärt werden.

- „MUST“, „REQUIRED“ und „SHALL“ bedeuten, dass die entsprechenden Definitionen unumgängliche Anforderungen der Spezifikation sind. Derart gekenn-

zeichnete Anforderungen müssen von einer BPEL-Engine erfüllt werden, wenn sie dem BPEL-Standard vollständig entsprechen soll. Es können Testfälle abgeleitet werden, die überprüfen, ob dies der Fall ist.

- „MUST NOT“ und „SHALL NOT“ sind absolute Verbote in der Spezifikation. Will eine BPEL-Engine dem BPEL-Standard entsprechen, dürfen die so gekennzeichneten Definitionen unter keinen Umständen vorkommen. Es können Testfälle abgeleitet werden, die überprüfen, ob dies der Fall ist.
- „SHOULD“ und „RECOMMENDED“ signalisieren, dass gewisse Eigenschaften unter gegebenen Umständen einen gewissen Punkt ignorieren dürfen. Bei entsprechenden Vorkommnissen müssen allerdings die kompletten Auswirkungen bekannt sein und vorsichtig abgewogen werden.
- „SHOULD NOT“ und „NOT RECOMMENDED“ signalisieren, dass gewisse Eigenschaften unter gegebenen Umständen existieren, in denen ein bestimmtes Verhalten akzeptabel oder sogar nützlich ist. Bei entsprechenden Vorkommnissen müssen allerdings die kompletten Auswirkungen bekannt sein und vorsichtig abgewogen werden.
- „MAY“ und „OPTIONAL“ bedeuten, dass ein Gegenstand wirklich optional ist. Ein so gekennzeichneter Gegenstand kann von einer BPEL-Engine erfüllt werden, muss aber nicht. Aus diesen Schlüsselbegriffen können folglich keine Testfälle für die Compliance Testsuite abgeleitet werden, da kein vorgegebenes Verhalten existiert.

3.3. Verfahren zur Ableitung von Testfällen

3.3.1. Einleitung

In diesem Abschnitt soll nun angegeben werden, wie konkrete Testfälle hergeleitet werden können. Dies geschieht unter Berücksichtigung der in Abschnitt 3.2 aufgelisteten Anforderungen mit Bezug auf die in Abschnitt 3.1 angegebenen Vorgehensweisen und Methoden zur Ableitung von Testfällen.

Dabei werden die angegebenen Vorgehensweisen gegebenenfalls, sofern dies sinnvoll ist, mit entsprechenden Vorgehensweisen ergänzt. Die ergänzenden Vorgehensweisen müssen die angegebenen Anforderungen an eine Compliance Testsuite erfüllen.

3.3.2. Probleme

Es ist nicht davon auszugehen, dass die direkte Herleitung der Testfälle und anschließende Erstellung der BPELUnit-Tests aus den hergeleiteten Testfällen aus den verfügbaren Ressourcen (BPEL-Standard / BPEL-Spezifikation, BPEL-Benutzer / Benutzerumfrage, Quelltext der Engines) ohne Komplikationen möglich ist. Tatsächlich sind Vorüberlegungen notwendig, um potentielle Probleme bei der Herleitung bereits im

Vorfeld zu erkennen.

Die Verbindung zwischen BPELUnit-Tests und der Überprüfung einer kompletten BPEL-Engine, wie es das Ziel der Compliance Testsuite ist, ist ein solches Problem.

BPELUnit-Tests sind Modultests für Webservices (siehe Abschnitt 2.3.1). Durch ihre Unabhängigkeit gegenüber den Engines bieten sie ein gutes Mittel, um diese Engines untereinander und gegeneinander zu testen.

Allerdings ist ihre Hauptaufgabe, wie schon erwähnt, das Testen einzelner Module. Dabei wird ein Prozess, der sogenannte PUT (process under test), getestet, indem ihm eine oder mehrere Nachrichten geschickt werden und die erhaltene Nachricht des PUT mit der erwarteten Nachricht verglichen wird.

Dahingegen ist eine BPEL-Engine kein Modul, das getestet werden kann. Vielmehr ist sie ein Server und Compiler zugleich.

Bevor ein Verfahren konkret angegeben werden kann, muss also geklärt werden, wie ein BPELUnit-Test so genutzt werden kann, dass er eine BPEL-Engine teilweise auf ihre Standardkompatibilität hin prüfen und somit in die Testsuite aufgenommen werden kann.

Dazu hilft die folgende Überlegung: Eine BPEL-Engine bekommt einen BPEL-Prozess, den sie bearbeiten soll. BPELUnit-Tests benötigen ebenfalls BPEL-Prozesse, die getestet werden können. Die Verbindung zwischen der Engine und BPELUnit sind folglich die Prozesse.

Mittels dieser Verbindung ist es BPELUnit möglich, eine BPEL-Engine anstatt eines BPEL-Prozesses zu testen. Dabei reicht es nicht, willkürlich Dummy-Prozesse zu erstellen. Letztendlich kann BPELUnit nur testen, was ein BPEL-Prozess zu leisten im Stande ist. Die Prozesse müssen also genau auf die Testfälle abgestimmt werden, damit die Testfälle eine BPEL-Engine überprüfen können.

Dabei dienen die BPEL-Prozesse als Stellvertreter der BPELUnit-Tests. Der Prozess wird auf der Engine deployt und mit Hilfe der BPELUnit-Tests wird überprüft, ob das Deployment korrekt war und ob sich der Prozess so verhält, wie er sich laut Standard verhalten soll.

Es muss folglich für jeden BPELUnit-Test und somit für jeden Testfall ein BPEL-Prozess erstellt werden, was bei der Angabe der Testfälle berücksichtigt werden muss.

Ein weiteres Problem besteht in der Nutzung des Quellcodes der BPEL-Engines. Die wenigsten BPEL-Engines besitzen einen frei zugänglichen Quellcode. Benutzt man nun den Quellcode der Engines, deren Quellcode zugänglich ist, so besteht die Gefahr, dass bei diesen Engines verhältnismäßig viele Fehler gefunden werden, was den Vergleich bezüglich den anderen Engines erschweren würde.

Da die Compliance Testsuite auch dazu verwendet werden soll, BPEL-Engines auf ihre Fähigkeiten hin zu prüfen und sich insbesondere anschließend für eine der Engines zu entscheiden, werden die quelloffenen Engines diesbezüglich sehr schnell benachteiligt.

Auf der anderen Seite ist es genau die Aufgabe der Compliance Testsuite eben diese Punkte aufzudecken. Sofern Inkompatibilitäten vorhanden sind, wäre es wünschenswert, diese auch aufzuspüren und anzugeben.

3.3.3. Angabe des Verfahrens

Nun soll das Verfahren angegeben werden. Dabei werden Punkte genannt, die beschreiben, wie Testfälle hergeleitet werden können.

Die Punkte sind nicht bezüglich des Ausführungszeitpunktes sortiert. Einige Punkte können parallel ausgeführt werden. In diesem Fall wird dies explizit angegeben.

1. Überprüfe für alle BPEL-Aktivitäten und BPEL-Elemente die semantische Definition des BPEL-Standards auf ihre Übersetzbarkeit in einen Testfall. Eine Definition ist nicht in einen Testfall übersetzbar, wenn sie mehrdeutig formuliert ist.
2. Erstelle für jede BPEL-Aktivität, die Punkt 1 erfüllt und jedes BPEL-Element, das Punkt 1 erfüllt, einen oder mehrere Testfälle, die die im Standard beschriebene Semantik überprüfen.
3. Führe parallel zu Schritt 1 und 2 eine Umfrage unter den BPEL-Engine Benutzern durch. Die Umfrage soll nach häufigen Problemen, bekannten Bugs oder aufgetretenen Kompatibilitätsproblemen zwischen den Engines fragen.
4. Wende nach Beendigung der Umfrage das unten angegebene Verfahren zur Auswertung der Umfrage an.
5. Durchsuche Datenbanken, die Defekte bezüglich der BPEL-Engines beinhalten und überprüfe die Defekte auf Übersetzbarkeit in einen Testfall und Relevanz. Wenn ein Defekt irrelevant ist, wird er nicht in einen Testfall umgesetzt, auch wenn dies möglich ist. Wann ein Defekt relevant ist, wird weiter unten angegeben.
6. Werden Eingabewerte für Testfälle benötigt, so ist das Äquivalenzklassenverfahren und optional das Grenzwertverfahren anzuwenden.

Die Umfrage soll wie folgt ausgewertet werden: Es muss zuerst überprüft werden, ob der angesprochene Defekt (Fehler, Inkompatibilität, Problem, o.Ä.) ein Defekt bezüglich des BPEL-Standards ist. Ist dies der Fall, erstelle einen oder mehrere Testfälle zu diesem Defekt. Ist dies nicht der Fall, ignoriere den angesprochenen Defekt.

Ein Defekt gilt mit Hinblick auf die Compliance Testsuite als relevant, wenn er sich auf die Kompatibilität der Engine bezüglich des Standards oder auf die Kompatibilität der Engines untereinander bezieht.

Im ersten Fall ist zu untersuchen, ob die Eigenschaft des Defekts vom BPEL-Standard gefordert wird. Wird sie nicht gefordert oder ist sie optional, ist der Defekt nicht relevant. Wie erkannt werden kann, ob eine Eigenschaft gefordert wird, wird in Abschnitt 3.2 beschrieben.

Im zweiten Fall muss ebenfalls überprüft werden, ob die Eigenschaften des Defekts

vom BPEL-Standard gefordert werden und ob sie optional sind. Werden sie nicht gefordert oder sind sie optional, so liegt das Problem der Inkompatibilität in der ungenauen Definition des BPEL-Standards. Somit ist der Defekt nicht relevant. Wie erkannt werden kann, ob eine Eigenschaft gefordert wird, wird in Abschnitt 3.2 beschrieben.

Eigenschaften, die mit „MAY“ gekennzeichnet sind, sind Beispiele für optionale Eigenschaften.

Die in Abschnitt 3.2 beschriebenen syntaktischen Anforderungen des BPEL-Standards an einen BPEL-Prozess lassen sich durch ein einfaches Erstellen eines entsprechenden BPEL-Prozesses realisieren.

Das führt dazu, dass für jede syntaktische Anforderung schlicht ein BPEL-Prozess erstellt und anschließend überprüft wird, ob der Prozess von der BPEL-Engine kompiliert wurde. Falls er kompiliert, unterstützt die BPEL-Engine offenbar die jeweilige Syntax.

Dies wiederum führt zu einer vielzähligen Erstellung einfacher BPEL-Prozesse, was nicht das Ziel dieser Bachelorarbeit sein sollte. Aus diesem Grund wird die Überprüfung syntatkischer Anforderungen des Standards nicht durchgeführt.

4. Priorisierung und Ableitung der Testfälle

4.1. Priorisierung in zwei Stufen

In diesem Kapitel sollen Testfälle mit Hilfe des angegebenen Verfahrens abgeleitet und priorisiert werden. Doch wann und wozu müssen die Testfälle priorisiert werden?

Die Ressourcen, aus denen Testfälle abgeleitet werden können und sollen, sind sehr umfangreich, da BPEL selbst ein komplexes Werkzeug ist und sich somit auch die BPEL-Engines komplex gestalten.

Allein die Spezifikation des BPEL-Standards umfasst viele hundert Seiten. Die Datenbanken, in denen Defekte aufgelistet werden, können sehr umfangreich sein, was von der Anzahl der Fehler einer BPEL-Engine abhängt. Dazu kommt, dass die Benutzer der BPEL-Engines möglicherweise viele Fehler umfangreich zu Protokoll geben. Diesbezüglich ist es möglich, dass sehr viel Aufwand für einen eventuell bereits erfassten Defekt betrieben wird.

Das Ziel möglichst viele Fehler auf diese Art abzudecken und mit der Compliance Testsuite zu finden überschreitet den Zeitrahmen einer Bachelorarbeit bei weitem. Der Grundgedanke dieser Bachelorarbeit ist vielmehr ein Verfahren präzise zu beschreiben und zu zeigen, dass dieses Verfahren zum Ziel führt. Dabei soll die Compliance Testsuite keinen maximalen Umfang erreichen, sondern gezielt einzelne Aspekte des Verfahrens aus mehreren Bereichen umsetzen. Aus diesem Grund muss das Gebiet klar abgesteckt und die potentiellen Testfälle priorisiert werden.

Um genau den eben erläuterten Eigenschaften gerecht zu werden, bietet sich eine Priorisierung in zwei Stufen an. Zuerst wird die Menge aller potentiellen Testfälle priorisiert. Entsprechend dieser Priorisierung werden anschließend Testfälle abgeleitet.

Die Menge der so erstellten Testfälle ist immer noch sehr groß. Um also dem Grundgedanken der Bachelorarbeit gerecht zu werden, müssen die Testfälle ein zweites Mal klar abgesteckt und priorisiert werden. Anschließend werden die entsprechenden Testfälle in BPELUnit-Tests umgesetzt und in die Compliance Testsuite integriert.

Wie die zwei Priorisierungen konkret ablaufen, wird in den Abschnitten 4.2 für die erste Priorisierung vor der Ableitung der Testfälle und 4.4 für die zweite Priorisierung nach der Ableitung der Testfälle genauer erklärt.

4.2. Priorisierung vor Ableitung der Testfälle

Die erste Priorisierung vor der Ableitung der Testfälle ist die schwierigere der beiden Priorisierungen. Dies liegt unter anderem daran, dass das Feld der potentiellen Testfälle umfangreich und weit gefächert ist.

Aus diesem Feld gilt es Aktivitäten zu priorisieren. Dabei muss man zwischen den einzelnen Ressourcen (Standard, Umfrage unter Benutzern der BPEL-Engines, Datenbanken für Defekte) unterscheiden. Zudem lassen sich die möglichen Testfälle und ihre Quellen durch Gewichtung und Zeitaufwand charakterisieren.

Gewichtung bezeichnet in diesem Fall die Schwere eines Testfalls bezüglich seiner Wichtigkeit gegenüber dem Standard. Ein Beispiel für unterschiedliche Gewichtungen sind die in Abschnitt 3.2 beschriebenen Schlüsselbegriffe, die ihrerseits eine Gewichtung der von ihnen gestellten Bedingungen vornehmen.

Als zeitaufwändig gelten Testfälle, die nur mit mehr Aufwand erstellt werden können als andere. Dies kann insbesondere der Fall sein, wenn Umfragen oder Datenbanken zur Ableitung verwendet werden, da die vorliegenden Informationen erst aufbereitet werden müssen, während die Informationen, die in der Spezifikation enthalten sind, oft schon in sehr guter Form für die Umsetzung in einen Testfall vorliegen.

Nun muss zwischen hoher Gewichtung und niedrigem Zeitaufwand abgewogen werden. Im optimalen Fall sollte angestrebt werden, beides zu erreichen.

Da die Spezifikation des BPEL-Standards ebenso stark gewichtet wie zeiteffizient ist, bietet sich diese Ressource besonders an. Aus diesem Grund soll der Schwerpunkt auf die Ableitung der Testfälle aus dem BPEL-Standard gelegt werden. Natürlich dürfen die anderen beiden Ressourcen nicht vernachlässigt werden.

Wie bereits erwähnt umfasst die Spezifikation des BPEL-Standards sehr viele Seiten. Daher ist es notwendig, die Bereiche der Spezifikation, in denen die Schritte 1 und 2 des in Unterabschnitt 3.3.3 angegebenen Verfahrens angewendet werden, klar abzustecken.

Klar ist, dass alle Kapitel der Spezifikation wichtige Informationen enthalten. Jede einzelne trägt zur Funktionsweise von BPEL bei. Aus diesem Grund habe ich mich vorerst auf die Kapitel 5, 6 und 7 der Spezifikation[3] konzentriert, wobei die Auswahl dieser Kapitel ohne besonderen Grund geschah.

Da die Aktivitäten ein sehr wichtiger Bestandteil von BPEL sind, hielt ich es bei meiner Wahl für notwendig, zumindest ein Kapitel zu berücksichtigen, das Aktivitäten behandelt. Dieser Entschluss entspricht auch der in Abschnitt 4.1 beschriebenen gewünschten Bereichsvielfalt.

Deshalb habe ich zusätzlich das Kapitel 10 der Spezifikation zur Ableitung von Testfällen herangezogen.

4.3. Ableitung der Testfälle

Nun sollen Testfälle korrekt abgeleitet und aufgelistet werden. In diesem Abschnitt wird zunächst die Struktur der Testfälle beschrieben. Anschließend werden Informationen angegeben, die dazu dienen, einen Testfall lesen zu können. Für jeweils einen Testfall aus den drei Quellen BPEL-Standard, Benutzerumfrage, Datenbank soll erklärt werden, wie dieser Testfall hergeleitet wurde. Zum Schluss werden alle hergeleiteten Testfälle angegeben.

Ein Testfall hat die folgende Struktur:

Testfall: <name>		Quelle: <name>
Beschreibung:	<beschreibung>	
Voraussetzungen:	<voraussetzungen>	
Durchführung:	<durchführung>	
Eingaben:	Eingabe	Erwartetes Ergebnis
	<eingabe 1>	<erwartetes ergebnis 1>
	<eingabe 2>	<erwartetes ergebnis 2>
	...	...

Jeder Testfall kann durch seinen *Namen* eindeutig identifiziert werden und gibt die Quelle an, aus der er abgeleitet wurde.

Quellen sind, wie schon mehrmals erwähnt, die Spezifikation des BPEL-Standards (Standard), die ausgewertete Umfrage unter Benutzern einer BPEL-Engine (Umfrage), sowie Datenbanken, die Defekte einer BPEL-Engine auflisten (Datenbank).

In der nächsten Zeile wird kurz *beschrieben*, aus welchem Aspekt sich der Testfall ergibt. Gleichzeitig sagt die Beschreibung viel über die Funktion des Testfalls aus.

Anschließend werden die *Voraussetzungen* angegeben, die erfüllt sein müssen, damit der Testfall durchgeführt werden kann. Da dies für die meisten Testfälle die gleichen Voraussetzungen sind, werden weiter unten zwei Standardvoraussetzungen definiert.

Nach den Voraussetzungen wird die *Durchführung* des Testfalls beschrieben. Zuerst wird eine allgemeine, gemeinsame Durchführung angegeben, die von den entsprechenden Eingabewerten unabhängig ist.

Danach wird die Durchführung bezüglich der unterschiedlichen *Eingabewerte* aufgeteilt. Unterschiede in der Durchführung bezüglich dieser Eingabewerte werden hier, ebenso wie das *erwartete Ergebnis*, angegeben.

Nun sollen die angekündigten Definitionen angegeben werden. Zuerst einmal lässt sich sagen, dass sich zwischen einem bereits deployten und einem noch nicht deployten BPEL-Prozess unterscheiden lässt.

Die Voraussetzung, dass ein BPEL-Prozess noch nicht deployt wurde, tritt insbesondere dann auf, wenn bestimmte Eigenschaften eines Prozesses von einer BPEL-Engine vor dem Deployment analysiert werden müssen und der Prozess im Fehlerfall nicht deployt werden darf.

Wenn die Aktivität eines BPEL-Prozesses während der Laufzeit überprüft werden soll, muss der Prozess zuerst erfolgreich deployt werden, was in diesem Fall als Voraussetzung aufgenommen wird.

In beiden Fällen muss der BPEL-Prozess, abgesehen von den im Testfall speziell angegebenen Eigenschaften, korrekt implementiert worden sein.

Es ergeben sich nun die zwei folgenden Definitionen:

- Standard-Voraussetzung 1: Der BPEL-Prozess wurde noch nicht auf dem Server deployt. Der BPEL-Prozess wurde korrekt implementiert. Das System befindet sich in fehlerfreiem Zustand.
- Standard-Voraussetzung 2: Der BPEL-Prozess wurde auf dem Server deployt und arbeitet korrekt. Der BPEL-Prozess wurde korrekt implementiert. Das System befindet sich in fehlerfreiem Zustand.

Der Begriff „fehlerfreier Zustand“ soll hier etwas genauer erklärt werden. In den meisten Fällen kann ein fehlerfreier Zustand, wie zum Beispiel ein fehlerhaftes Betriebssystem oder fehlerhafte Hardware, nicht garantiert werden. Es ist lediglich gemeint, dass die erstellten Testfälle, die eine derartige Voraussetzung fordern, eventuell in einem fehlerbehafteten System nicht funktionieren. Sollten betreffende Situationen bekannt sein, ist dies zu berücksichtigen.

Die Compliance Testsuite stellt also keinen Anspruch korrekt zu funktionieren, wenn die Umgebung nicht korrekt funktioniert. Dies ist zum Beispiel der Fall, wenn Komponenten wie der Server oder die BPEL-Engine nicht korrekt installiert sind oder Hardware wie CPU und Hauptspeicher die Daten falsch berechnen oder speichern.

Ein BPEL-Prozess arbeitet korrekt, wenn alle Eigenschaften des Prozesses, mit Ausnahme der gewünschten, erwarteten Fehler, korrekt arbeiten. Man kann davon ausgehen, dass sie dies tun, indem sie entweder durch andere Tests überprüft werden oder dem BPEL-Standard exakt entsprechen. Damit ist gemeint, dass die Eigenschaften des Prozesses keine syntaktischen Fehler enthalten, keine Kompilierungsfehler verursachen und gegebenenfalls das Ergebnis zurückliefern, welches von ihnen erwartet wird.

Des weiteren sollen die folgenden Begriffe erklärt werden:

Ist in einem Testfall vom Setzen von Attributen die Rede, so sollen die Attribute auf einen beliebigen, festen, gültigen Wert gesetzt werden.

Werden optionale Attribute gesetzt, bedeutet dies, dass die Attribute vorhanden sind. Wird das Setzen eines optionalen Attributes nicht erwähnt, soll das Attribut nicht vorhanden sein.

Sollen Aktivitäten oder Elemente enthalten sein, müssen diese Elemente der Spezifikation des BPEL-Standards entsprechen.

Die Standardfehler werden in Anhang B aufgelistet. Die Erklärung eines jeden einzelnen Fehlers würde dabei zu weit führen und ist in den meisten Fällen bereits anhand des Namens ersichtlich. Für weiterführende Informationen kann die Spezifikation des

BPEL-Standards [3] benutzt werden.

Als Startaktivität wird eine *receive*- oder *pick*-Aktivität bezeichnet, die das *createInstance*-Attribut auf *yes* gesetzt hat.

Zum Schluss sollte noch erwähnt werden, dass zur Durchführung der Testfälle zusätzlich das folgende gehört:

- Fall Standard-Voraussetzung 1: Es wird versucht, den BPEL-Prozess auf der BPEL-Engine zu deployen. In den meisten Fällen wird danach lediglich überprüft, ob das Deployment erfolgreich war.
- Fall Standard-Voraussetzung 2: Die zu überprüfende Tätigkeit wird von BPELUnit angestoßen und anschließend überprüft, ob sie dem erwarteten Ergebnis entspricht.

Nun soll exemplarisch ein Testfall aus dem BPEL-Standard abgeleitet werden. Für diesen Zweck wird der Testfall „*portType*-Attribut bei *receive*-Aktivität“ gewählt. Die Aussage, aus der der Testfall abgeleitet wurde, befindet sich in Kapitel 5.2 der Spezifikation des BPEL-Standards [3] und ist mit „SA00005“ gekennzeichnet. Konkret lautet die Aussage: „If the portType attribute is included for readability, the value of the portType attribute MUST match the portType value implied by the combination of the specified partnerLink and the role implicitly specified by the activity“.

Da in der Aussage ein „MUST“ enthalten ist, kann man erkennen, dass die Eigenschaft ohne Ausnahme gefordert ist. Zudem lässt sich die Aussage fast ohne Änderung in einen Testfall umsetzen und ist daher umsetzbar. Punkt 1 des Verfahrens ist also erfüllt.

Nun gilt es Punkt 2 des Verfahrens umzusetzen. Da für den Testfall Eingabewerte benötigt werden, ist gemäß Punkt 6 das Äquivalenzklassenverfahren anzuwenden. Offenbar ist der Parameter das *portType*-Attribut. Für diesen Parameter lassen sich zwei Äquivalenzklassen bilden.

Die erste Äquivalenzklasse enthält alle Werte, die einen Fehler erzeugen, die zweite Äquivalenzklasse enthält alle Werte, die keinen Fehler erzeugen. Die erste Äquivalenzklasse enthält unendlich viele Elemente, da die Zeichenkette beliebig lang gewählt werden kann. Die zweite Äquivalenzklassen enthält dagegen lediglich ein einziges Element, da dieses den gleichen Wert besitzen muss, wie das des *portType*-Attributs des zugehörigen *partnerLink*-Elements, welches fest gewählt wurde.

Da die beiden Äquivalenzklassen entweder keine klaren Grenzen besitzen oder aber nur ein Element enthalten, lässt sich das Grenzwertverfahren nicht anwenden.

Es ergibt sich der unten angegebene Testfall.

Für die Datenbank-Ressource soll ebenfalls ein Testfall exemplarisch hergeleitet werden. Es wird der Testfall „Auswirkung der *fromPart*-Elemente auf das *variable*-Attribut einer *receive*-Aktivität“ gewählt. Für diesen Zweck wird die Liste bekannter Fehler der Apache ODE BPEL-Engine benutzt [4]. Dort wird unter dem Punkt „receive“ die Aussage „ODE does not yet support the <fromPart> syntax. Therefore, the variable attribute must be used.“ gemacht.

Diese Aussage ist bezüglich des Standards relevant (vgl. „SA00055“ der Spezifikation des BPEL-Standards). Es ist ebenfalls leicht ersichtlich, dass sie in einen Testfall umsetzbar ist, da sie sich ebenfalls fast ohne Änderungen übernehmen lässt. Folglich ist Punkt 5 des Verfahrens erfüllt und die Aussage kann in einen Testfall umgewandelt werden.

Für diesen Testfall werden keine Eingabewerte benötigt, da es sich um die Frage des Vorhandenseins eines Attributs handelt.

Es ergibt sich der weiter unten angegebene Testfall.

An dieser Stelle ist zu sagen, dass die angestrebten Umfragen keine brauchbaren Ergebnisse erzielt haben. Dies hat mehrere Gründe, welche kurz erläutert werden sollen.

Die Idee, Umfragen in den Foren der Entwickler der BPEL-Engines zu stellen schlug durch die Tatsache fehl, dass die Foren keine entsprechenenden Bereiche aufwiesen. Ein Forum bot die Registrierung sogar ausschließlich für Firmen an.

Die zweite Idee, entsprechende Xing-Gruppen zu finden, war ebenfalls nicht von Erfolg gekrönt, da es keine Gruppe gab, die sich eindeutig zu BPEL zuordnen lies.

Einzig die ODE-Mailing-List bot eine Möglichkeit eine Umfrage durchzuführen. Leider hat sich dort niemand mit brauchbaren Vorschlägen oder Fehlern gemeldet. Es wurde auf die Fehlerdatenbanken verwiesen, die aber bereits zu einem anderen Verfahren gehören.

Aus diesen Gründen konnten keine Testfälle aus Umfragen gewonnen werden. Die Umfrage wurde aus dem Verfahren herausgenommen, da der Aufwand im Verhältnis zum Nutzen zu hoch ist.

Folgend werden die abgeleiteten Testfälle angegeben. Da sich einige Testfälle nur in der betreffenden Aktivität unterscheiden, werden hier nicht alle Testfälle aufgeführt. Die Testfälle, die hier nicht angegeben werden, werden in Anhang A aufgelistet.

Testfall: Anforderung an *fault handler*-Element in BPEL-Prozess		Quelle: Standard
Beschreibung:	Ein BPEL-Prozess kann das Attribut *exitOnStandardFault* enthalten. Ist es auf *yes* gesetzt, darf kein *fault handler*-Element benutzt werden, der einen Standardfehler abfängt.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess, der das *exitOnStandard-Fault*-Attribut gesetzt hat. Erstelle ein *fault handler*-Element des BPEL-Prozesses.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Das *exitOnStandardFault*-Attribut wird auf *yes* gesetzt. Das *fault handler*-Element wird derart erstellt, dass es einen Standard Fehler abfängt.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Das *exitOnStandardFault*-Attribut wird auf *yes* gesetzt. Das *fault handler*-Element wird derart erstellt, dass es keinen Standard Fehler abfängt.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.
	Das *exitOnStandardFault*-Attribut wird auf *no* gesetzt. Das *fault handler*-Element wird derart erstellt, dass es einen Standard Fehler abfängt.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.
	Das *exitOnStandardFault*-Attribut wird auf *no* gesetzt. Der *fault handler*-Element wird derart erstellt, dass es keinen Standard Fehler abfängt.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: *portType*-Attribut bei *receive*-Aktivität		Quelle: Standard
Beschreibung:	Das *portType*-Attribut der *receive*-Aktivität ist optional. Wird es angegeben, muss es mit dem Wert des *portType*-Attributs des entsprechenden *partnerLink*-Elements und dem damit zusammenhängenden *role*-Element übereinstimmen.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess mit einer *receive*-Aktivität und setze ihr *portType*-Attribut.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Das *portType*-Attribut soll dabei einen anderen Wert bekommen, als der Wert des zugehörigen *partnerLink*-Elements und dem damit zusammenhängenden *role*-Element.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Das *portType*-Attribut soll dabei den gleichen Wert bekommen, wie der Wert des zugehörigen *partnerLink*-Elements und dem damit zusammenhängenden *role*-Element.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: *portType*-Attribut bei *reply*-Aktivität		Quelle: Standard
Beschreibung:	Das *portType*-Attribut der *reply*-Aktivität ist optional. Wird es angegeben, muss es mit dem Wert des *portType*-Attributs des entsprechenden *partnerLink*-Elements und dem damit zusammenhängenden *role*-Element übereinstimmen.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess mit einer *reply*-Aktivität und setze ihr *portType*-Attribut.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Das *portType*-Attribut soll dabei einen anderen Wert bekommen, als der Wert des zugehörigen *partnerLink*-Elements und dem damit zusammenhängenden *role*-Element.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Das *portType*-Attribut soll dabei den gleichen Wert bekommen, wie der Wert des zugehörigen *partnerLink*-Elements und dem damit zusammenhängenden *role*-Element.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: *for* und *until* bei *wait*-Aktivität		Quelle: Standard
Beschreibung:	Die *wait*-Aktivität muss genau eines der beiden Aktivierungskriterien *for* und *until* enthalten.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess mit einer *wait*-Aktivität.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Die *wait*-Aktivität soll keines der beiden Aktivierungskriterien *for* und *until* enthalten.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Die *wait*-Aktivität soll das Aktivierungskriterium *for*, nicht aber das Aktivierungskriterium *until* enthalten.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.
	Die *wait*-Aktivität soll das Aktivierungskriterium *until*, nicht aber das Aktivierungskriterium *for* enthalten.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.
	Die *wait*-Aktivität soll die beiden Aktivierungskriterien *for* und *until* enthalten.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.

Testfall: Aufruf der *compensateScope*-Aktivität		Quelle: Standard
Beschreibung:	Die *compensateScope*-Aktivität darf nur aus einem *fault handler*-Element, einem anderen *compensation handler*-Element oder einem *termination handler*-Element heraus aufgerufen werden.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess mit einer *compensateScope*-Aktivität und rufe die *compensateScope*-Aktivität in diesem BPEL-Prozess auf.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Die *compensateScope*-Aktivität soll dabei nicht aus einem *fault handler*-Element, einem anderen *compensation handler*-Element oder einem *termination handler*-Element heraus aufgerufen werden.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Die *compensateScope*-Aktivität soll dabei aus einem *fault handler*-Element heraus aufgerufen werden.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.
	Die *compensateScope*-Aktivität soll dabei aus einem anderen *compensation handler*-Element als dem *fault handler*-Element heraus aufgerufen werden.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.
	Die *compensateScope*-Aktivität soll dabei aus einem *termination handler*-Element heraus aufgerufen werden.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: Aufruf der *rethrow*-Aktivität		Quelle: Standard
Beschreibung:	Die *rethrow*-Aktivität darf nur in einem *fault handler*-Element benutzt werden.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess mit einer *rethrow*-Aktivität und rufe die *rethrow*-Aktivität auf.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Die *rethrow*-Aktivität soll von außerhalb eines *fault handler*-Elements aufgerufen werden.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Die *rethrow*-Aktivität soll aus einem *fault handler*-Element heraus aufgerufen werden.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: Attribute im *partnerLink*-Element		Quelle: Standard
Beschreibung:	In einer *partnerLink*-Deklaration müssen entweder die Attribute *myRole*, *partnerRole* oder beide angegeben werden.	
Voraussetzungen:	Standard-Vorausetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess, in dem ein *partnerLink*-Element definiert wird. Erstelle zudem ein zugehöriges *partnerLinkType*-Element in der entsprechenden WSDL-Datei.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Weder das *myRole*-Attribut noch das *partnerRole*-Attribut sollen gesetzt werden.	Der BPEL-Prozess darf nicht kompilieren.
	Es soll das *myRole*-Attribut, nicht aber das *partnerRole*-Attribut gesetzt werden.	Der BPEL-Prozess muss kompilieren.
	Es soll das *partnerRole*-Attribut, nicht aber das *myRole*-Attribut gesetzt werden.	Der BPEL-Prozess muss kompilieren.
	Es sollen sowohl das *myRole*-Attribut als auch das *partnerRole*-Attribut gesetzt werden.	Der BPEL-Prozess muss kompilieren.

Testfall: Einzigartigkeit des Namens eines *partnerLink*-Elements in einem BPEL-Prozess		Quelle: Standard
Beschreibung:	Ein *partnerLink*-Element kann in einem BPEL-Prozess angegeben werden. Wird es dort angegeben, darf sein Name von keinen anderen *partnerLink*-Elementen desselben BPEL-Prozesses verwendet werden.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess und erzeuge zwei *partnerLink*-Elemente. Erstelle zudem zwei zugehörige *partnerLinkType*-Elemente in der entsprechenden WSDL-Datei.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Setze das *name*-Attribut beider *partnerLink*-Elemente auf denselben Wert.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Setze das *name*-Attribut des ersten *partnerLink*-Elements auf einen vom *name*-Attribut des zweiten *partnerLink*-Elements verschiedenen Wert.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: *type*- und *element*-Attribute in *property*-Definition		Quelle: Standard
Beschreibung:	Eine *property*-Definition kann als Attribut entweder *type* oder *element* enthalten, aber nicht beides gleichzeitig.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle eine WSDL-Datei mit einer *property*-Definition.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Die *property*-Definition hat sowohl das *type*-Attribut als auch das *element*-Attribut gesetzt.	Die BPEL-Engine darf die WSDL-Datei nicht kompilieren.
	Die *property*-Definition hat weder das *type*-Attribut noch das *element*-Attribut gesetzt.	Die BPEL-Engine darf die WSDL-Datei nicht kompilieren.
	Die *property*-Definition hat das *type*-Attribut gesetzt, nicht aber das *element*-Attribut.	Die BPEL-Engine muss die WSDL-Datei kompilieren.
	Die *property*-Definition hat das *element*-Attribut gesetzt, nicht aber das *type*-Attribut.	Die BPEL-Engine muss die WSDL-Datei kompilieren.

Testfall: Beziehung zwischen den *toPart*-Elementen der *invoke*-Aktivität und den *part*-Elementen der *message*-Definition		Quelle: Standard
Beschreibung:	Ist in einer *invoke*-Aktivität das *toParts*-Element vorhanden, muss für jedes *part*-Element der zugehörigen *message*-Definition ein *toPart*-Element vorhanden sein.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess, der eine *invoke*-Aktivität enthält. Erstelle weiterhin eine entsprechende *message*-Definition, welche zwei *part*-Elemente enthält.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Die *invoke*-Aktivität soll das *toParts*-Element enthalten. Dabei soll nur ein *toPart*-Element entsprechend eines der beiden *part*-Elemente der *message*-Definition angegeben werden.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Die *invoke*-Aktivität soll das *toParts*-Element enthalten. Dabei sollen zwei *toPart*-Elemente entsprechend der *part*-Elemente der *message*-Definition angegeben werden.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: Auswirkung der *fromPart*-Elemente auf das *variable*-Attribut einer *receive*-Aktivität		Quelle: Standard / Datenbank
Beschreibung:	Enthält eine *receive*-Aktivität *fromPart*-Elemente, darf das *variable*-Attribut nicht gesetzt sein.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess, der eine *receive*-Aktivität enthält, in der *fromPart*-Elemente vorhanden sind.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Setze das *variable*-Attribut der *receive*-Aktivität.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	In der *receive*-Aktivität soll das *variable*-Attribut nicht gesetzt sein.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: Struktur der *receive*-Aktivität bei fehlenden *part*-Elementen der *message*-Definition		Quelle: Standard
Beschreibung:	Eine *receive*-Aktivität empfängt von anderen Webservices Nachrichten. Wenn die zugehörige *message*-Definition einer solchen Nachricht keine *part*-Elemente enthält, müssen die *fromParts*-Elemente der *receive*-Aktivität weggelassen werden.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess, der eine *receive*-Aktivität enthält. Erstelle des weiteren eine entsprechende *message*-Definition. Die *message*-Definition soll dabei keine *part*-Elemente enthalten.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Die *receive*-Aktivität soll mindestens ein gültiges *fromParts*-Element enthalten.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Die *receive*-Aktivität soll kein *fromParts*-Element beinhalten.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: Struktur der *reply*-Aktivität bei fehlenden *part*-Elementen der *message*-Definition		Quelle: Standard
Beschreibung:	Eine *reply*-Aktivität schickt Nachrichten an andere Webservices. Wenn die zugehörige *message*-Definition einer solchen Nachricht keine *part*-Elemente enthält, müssen die *toParts*- Elemente der *reply*-Aktivität weggelassen werden.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess, der eine *reply*-Aktivität enthält. Erstelle des weiteren eine entsprechende *message*-Definition. Die *message*-Definition soll dabei keine *part*-Elemente enthalten.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Die *reply*-Aktivität soll mindestens ein gültiges *toParts*-Element enthalten.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Die *reply*-Aktivität soll kein *toParts*-Element beinhalten.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: Auswirkung der *toPart*-Elemente auf das *variable*-Attribut einer *reply*-Aktivität		Quelle: Standard / Datenbank
Beschreibung:	Enthält eine *reply*-Aktivität *toPart*-Elemente, darf das *variable*-Attribut nicht gesetzt sein.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess, der eine *reply*-Aktivität enthält, in der *toPart*-Elemente vorhanden sind.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Setze das *variable*-Attribut der *reply*-Aktivität.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	In der *reply*-Aktivität soll das *variable*-Attribut nicht gesetzt sein.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: Startaktivitäten in einem *extensionActivity*-Element		Quelle: Standard
Beschreibung:	Ein *extensionActivity*-Element muss mindestens eine Startaktivität enthalten.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess, der ein *extensionActivity*-Element enthält und gib dem Element eine Aktivität.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Bei der Aktivität handelt es sich um keine Startaktivität.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Bei der Aktivität handelt es sich um eine Startaktivität.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: Synchrone und asynchrone BPEL-Prozesse		Quelle: Datenbank
Beschreibung:	Eine BPEL-Engine muss sowohl synchrone als auch asynchrone BPEL-Prozesse kompilieren können.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess, der eine beliebige Aktivität enthält.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Bei dem erstellten BPEL-Prozess soll es sich um einen synchronen BPEL-Prozess handeln.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.
	Bei dem erstellten BPEL-Prozess soll es sich um einen asynchronen BPEL-Prozess handeln.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: Benutzung des *fromParts*-Elements in der *pick*-Aktivität		Quelle: Datenbank
Beschreibung:	Die *pick*-Aktivität muss das *fromParts*-Element anstelle des *variable*-Attributs akzeptieren.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess, der eine *pick*-Aktivität enthält. Die *pick*-Aktivität soll ein *fromParts*-Element und kein *variable*-Attribut enthalten.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Keine weiteren Eingaben benötigt.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: Kompilierung der *validate*-Aktivität		Quelle: Datenbank
Beschreibung:	Das Benutzen der *validate*-Aktivität in einem BPEL-Prozess muss möglich sein.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess, der eine *validate*-Aktivität enthält.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Keine weiteren Eingaben benötigt.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

4.4. Priorisierung nach Ableitung der Testfälle

Aus den soeben hergeleiteten Testfällen können nun BPELUnit-Tests erstellt werden. Die Anzahl der Testfälle ist mittlerweile überschaubar geworden. Allerdings müssen sie, wie in Abschnitt 4.1 erklärt, noch einmal priorisiert werden.

Die Testfälle, die die höchste Priorität bekommen, werden anschließend in BPELUnit-Tests umgesetzt. Die genaue Umsetzung wird im nachfolgenden Abschnitt beschrieben.

Die Priorisierung soll so durchgeführt werden, dass eine klar umrissene Menge von Testfällen ausgewählt und implementiert wird. Wichtig hierbei ist, dass die Elemente der Menge, also die ausgewählten Testfälle, nicht willkürlich ausgewählt werden sollen.

Vielmehr sollen Ansprüche an eine solche Menge gestellt werden, wobei der Menge anschließend alle Testfälle zugeordnet werden, die diesen Ansprüchen gerecht werden. Es werden also keine Testfälle, sondern Ansprüche an eine Menge von Testfällen ausgewählt.

Wie bereits erwähnt soll die Menge der Testfälle klar umrissen sein. Dies bedeutet, dass eine oder mehrere Aktivitäten konkret benannt werden müssen und dann ausschließlich alle Testfälle, die sich auf diese Aktivität beziehen, implementiert werden sollen.

Die ausgewählten Aktivitäten sollten dabei ein möglichst breit gefächertes Feld an Eigenschaften abdecken. Mit Eigenschaften sind Forderungen eines Testfalls in bestimmten Teilgebieten, wie zum Beispiel das Vorhandensein von Attributen oder der Bezug auf die WSDL-Datei gemeint. Diese Entscheidung ist sinnvoll, da so ein guter, exemplarischer Überblick gewährleistet werden kann.

Betrachtet man die abgeleiteten Testfälle, so gibt es mehrere Möglichkeiten, die eben angegebenen Eigenschaften zu erfüllen. Ich habe mich entschieden, die *receive*- und *reply*-Aktivitäten zu implementieren.

Diese beiden Aktivitäten decken ein breit gefächertes Feld an Eigenschaften ab. Es sind Testfälle dabei, die auf Attribute hin prüfen, die sich auf Wechselwirkungen zwischen Attributen oder Elementen beziehen und die eine Verbindung zur WSDL-Datei herstellen.

Außerdem bezieht sich ein Testfall konkret auf das Testen von synchronen und asynchronen BPEL-Prozessen, gibt aber keine bestimmte Aktivität an. Er wird ebenfalls der Menge hinzugefügt, wobei die beliebige Aktivität als *receive*- und *reply*-Aktivität definiert wird.

Dieser Testfall erweitert das Feld noch einmal um eine neue Eigenschaft und ist somit entsprechend den gestellten Forderungen gut für die Implementierung geeignet.

4.5. Erstellung der BPELUnit-Tests

4.5.1. Distanzierung von BPELUnit

Nun ist es an der Zeit, die Testfälle in BPELUnit-Tests umzusetzen und zu automatisieren. Dabei werden, wie schon mehrmals beschrieben, BPEL Prozesse gemäß den Vorgaben der Testfälle erstellt, deployt und gegebenenfalls mit BPELUnit auf ihre Korrektheit hin überprüft.

Bei genauerer Betrachtung von BPELUnit fällt auf, dass ausschließlich deployte BPEL-Prozesse von BPELUnit getestet werden können. BPELUnit ist nicht in der Lage, BPEL-Prozesse zu deployen und anschließend zu überprüfen, ob das Deployment erfolgreich war.

Da allerdings, wie schon in Abschnitt 4.2 erwähnt, bewusst schwerwiegende Testfälle gewählt wurden, ist leicht zu verstehen, dass die Fehlererkennung bezüglich dieser Punkte bereits vor dem Deployment von der BPEL-Engine durchgeführt werden muss. Dies wird vom BPEL-Standard gefordert, in dem das Deployment eines derart falschen BPEL-Prozesses verboten wird.

Nun beziehen sich alle Testfälle, die implementiert werden sollen, auf die Frage des Deployments. Eine entsprechende Implementierung kann also nicht mit BPELUnit umgesetzt werden. Aus diesem Grund weiche ich von der Aufgabenstellung ab und verwende BPELUnit nicht zur Implementierung der Testfälle.

Dieser Aspekt ist erst deutlich geworden, nachdem die Testfälle priorisiert und erstellt worden sind. Erst zu diesem Zeitpunkt wurde klar, dass sich die Testfälle auf die Frage des Deployments beziehen und dass BPELUnit nicht in der Lage ist, derartige Testfälle umzusetzen.

Dennoch ist die Entscheidung sich von BPELUnit zu distanzieren meiner Auffassung nach richtig, da die Alternative darin besteht, Testfälle geringerer Wichtigkeit zu erstellen, die erst nach dem Deployment überprüft werden müssen. Die Testfälle, die in die Compliance Testsuite aufgenommen werden, sollten allerdings eine maximale Bedeutsamkeit besitzen.

Um das Deployment und die anschließende Überprüfung durchzuführen, soll alternativ ein automatisiertes Skript verwendet werden. Wie dies genau funktioniert, soll im nachfolgenden Abschnitt genauer erläutert werden.

4.5.2. Lösung mit Apache Ant

Um das Deployment der BPEL-Prozesse und die anschließende Überprüfung des Deployments durchzuführen, soll anstelle von BPELUnit ein automatisiertes Skript verwendet werden. Da Apache Ant bereits zur Automatisierung der BPELUnit-Tests laut Aufgabenstellung verwendet werden sollte, bietet es sich auch für diese Aufgabe an.

Es muss also für jede BPEL-Engine ein Test-Skript erstellt werden, das sich um die

Überprüfung des Deployments entsprechender BPEL-Prozesse kümmert, sowie ein Haupt-Skript, welches die Test-Skripte nacheinander ausführt.

Die konkrete Implementierung des Ant-Skripts soll an dieser Stelle nicht schriftlich angegeben werden, da der Quelltext der BPEL-Prozesse und des Skripts sehr lang ist. Zudem unterscheiden sich die BPEL-Prozesse, bedingt durch die Struktur der Sprache, oft nur an bestimmten Stellen voneinander, wodurch immer wieder eine Wiederholung größerer Passagen vorkommt.

Aus diesem Grund liegen das Ant-Skript und sämtliche BPEL-Prozesse nur in elektronischer Form auf der der Arbeit beigelegten CD vor.

Das Ant-Skript für eine BPEL-Engine arbeitet wie folgt:

1. Der Server wird gestartet.
2. Es wird versucht, alle BPEL-Prozesse zu deployen, indem entsprechende Dateien in ein entsprechendes Verzeichnis des Servers kopiert werden. Der Server erkennt die neuen Dateien und bearbeitet sie.
3. Die HTML-Seite der jeweiligen BPEL-Engine, auf der die erfolgreich deployten BPEL-Prozesse aufgelistet werden, wird von einem Parser überprüft.
4. Die Ergebnisse werden in einer Datei gespeichert.

Am Ende liegt für jede BPEL-Engine eine Datei vor, die die Ergebnisse des Tests enthält.

5. Evaluation der Compliance Testsuite

5.1. Testdurchführung

In diesem Kapitel wird die Apache ODE BPEL-Engine durch die Compliance Testsuite getestet. Dies erlaubt es, Rückschlüsse auf den Grad der Implementierung des BPEL-Standards zu ziehen und dient zur Evaluation der Testsuite.

Allerdings ist zu bemerken, dass die Compliance Testsuite hinsichtlich des BPEL-Standards keinesfalls vollständig ist. Deshalb sollte bedacht werden, dass die Testergebnisse nicht sehr aussagekräftig sind und auch keine Bewertung der BPEL-Engine vornehmen können und dürfen.

Es ist durchaus möglich, dass das kleine Fenster der möglichen Testfälle, welches von der Compliance Testsuite überprüft wird, die zu testende Engine im Vergleich zu anderen Engines, die möglicherweise durch die Compliance Testsuite getestet werden, in einem besonders schlechten oder guten Licht dastehen lässt. Würde man dahingegen alle Testfälle berücksichtigen, wäre es allerdings denkbar, dass sich die Situation umdreht.

5.2. Probleme bei der Wahl einer zweiten BPEL-Engine

Die Aufgabenstellung dieser Bachelorarbeit sieht vor, dass die Apache ODE BPEL-Engine und eine beliebige zweite BPEL-Engine durch die Compliance Testsuite getestet werden sollen.

Apache ODE bereitete hierbei keine Probleme und lies sich nahezu problemlos installieren und testen.

Nun musste eine zweite BPEL-Engine gewählt werden. Bei der Wahl fielen die kostenpflichtigen BPEL-Engines verständlicherweise weg. Das Feld der verbliebenen, kostenlosen Kandidaten fiel sehr klein aus. Zudem hatte ich vor, eine BPEL-Engine zu wählen, die den BPEL-Standard 2.0 unterstützen soll und aktiv weiterentwickelt wird, wodurch die Auswahl sehr überschaubar wurde. Konkret standen ActiveBPEL, jBPM und GlassFish zur Auswahl.

Zuerst habe ich die ActiveBPEL BPEL-Engine gewählt. Nach Erstellung der entsprechenden Dateien, die für das Deployment auf der Engine notwendig waren, wurden diverse Fehlermeldungen ausgegeben, die sich nur durch eine Veränderung der WSDL-Datei beheben lassen würden.

Dies ist jedoch nicht im Interesse der Compliance Testsuite, da die Interoperabilität

der BPEL-Prozesse sichergestellt werden soll (vgl. Abschnitt 1.2, sowie Aufgabenstellung). Folglich dürfte kein einziger Testfall der Compliance Testsuite erfolgreich abschließen.

Da dies bezüglich der Aufgabenstellung nicht akzeptabel war, habe ich als zweites die BPEL-Engine jBPM ausprobiert. Nach diversen großen Problemen, die in Abschnitt 6.1 genauer erklärt werden, wurde ersichtlich, dass der Application Server JBoss, auf dem die BPEL-Engine in dem Fall deployt werden muss, nicht in der Lage war, die deployten Prozesse anzuzeigen.

Dies kann mehrere Ursachen haben. Es könnte beispielsweise sein, dass der Server nicht korrekt installiert oder die BPEL-Prozesse nicht korrekt deployt wurden. Letztendlich konnte das Deployment der BPEL-Prozesse nicht überprüft werden.

Aus diesen Gründen war es mir nicht möglich, eine zweite BPEL-Engine durch die Compliance Testsuite mit Erfolg zu testen. Es soll dennoch das Ant-Skript für jBPM soweit wie möglich erstellt werden. Die Funktionsweise des Skripts wird in Kaptitel 5.4 erklärt.

5.3. Apache ODE BPEL-Engine im Test

Zuerst wird die Apache ODE BPEL-Engine durch die Compliance Testsuite getestet. Dabei werden insgesamt 14 Tests durchgeführt. Um die Lesbarkeit zu erhöhen und den Platzbedarf in Grenzen zu halten, werden die Namen der Testfälle abgekürzt. Die Abkürzungen sollten dennoch Rückschlüsse auf den entsprechenden Testfall zulassen und werden in Anhang C aufgelistet. Eine kurze Übersicht der Testergebnisse wird in Anhang D gegeben.

Die Tests zeigen einige interessante Ergebnisse. Zuerst lässt sich am Testfall „Sync-Prozess“ erkennen, dass Apache ODE zwar synchrone, jedoch keine asynchronen BPEL-Prozesse unterstützt. Dies war zu erwarten, da der Testfall aus genau dieser Tatsache resultiert.

An den restlichen Testfällen lassen sich nun drei unterschiedliche Vorgänge erkennen. Zum einen wurden zwei Testreihen fehlerfrei abgeschlossen. Zum anderen gab es bei zwei Testreihen Fehler. Zusätzlich wird hier jedoch ein Fall deutlich, bei dem zwei Testreihen durch die Bauart der Testfälle begünstigt wurden.

Die Testfälle „PortTypeReceive“ und „PortTypeReply“, laufen erfolgreich durch. Apache ODE hat die Situationen korrekt erkannt und deployt die fehlerhaften BPEL-Prozesse nicht.

Die Testfälle „FromPartReceive“ und „ToPartReply“ schlagen alle fehl. Dies liegt daran, dass Apache ODE die *fromPart*- und *toPart*-Elemente nicht unterstützt.

Die interessanten Testfälle sind allerdings „StrukturReceive“, sowie „StrukturReply“. Wie wir nun wissen, werden die *fromPart*- und *toPart*-Elemente von Apache ODE nicht unterstützt. Dennoch schlägt der Test nicht fehl.

Dies liegt am Aufbau der Testfälle. Es wird gefordert, dass die *message*-Definition keine *part*-Elemente enthält. Somit wird für den Testfall ein BPEL-Prozess mit einem *fromPart*- bzw. *toPart*-Element erstellt und einer ohne.

Zufällig ist der Testfall, der als Ergebnis die Kompilierung erwartet, derjenige, der kein *fromPart*- bzw. *toPart*-Element enthält. Apache ODE hat also keine Probleme ihn zu kompilieren.

Von dem Testfall, der diese Elemente enthält, wird allerdings erwartet, dass er nicht kompiliert. Dies tut er auch nicht, was allerdings nicht an der ursprünglichen Absicht des Testfalls liegt, sondern vielmehr an dem bekannten Fehler, dass Apache ODE keine *fromPart*- bzw. *toPart*-Elemente unterstützt.

Hierbei kann man erkennen, dass Fehler, die in einem Testfall nicht getestet werden sollen, den Testfall dennoch beeinflussen können. In diesem Fall wurde der Testfall somit zufällig als bestanden gekennzeichnet.

Hätte man nun die Testfälle „StrukturReceive“ und „StrukturReply“ so verfasst, dass das Vorhandensein eines *fromPart*- bzw. *toPart*-Elements gefordert wird und die Eigenschaft der *message*-Definition *part*-Elemente zu besitzen variabel wäre, hätten die beiden Tests, die eine Kompilierung fordern, offenbar fehlgeschlagen.

Der Test der Apache ODE BPEL-Engine zeigte interessante Ergebnisse. Es wurden Tests erfolgreich ausgeführt, schlugen fehl und wurden zufällig als erfolgreich gekennzeichnet. Insgesamt wurden 14 Tests durchgeführt, von denen 9 erfolgreich abgeschlossen wurden.

Die Tatsache, dass Testfälle zufällig als erfolgreich gekennzeichnet wurden, ist eine nicht wünschenswerte Eigenschaft, da im Allgemeinen Fehler dadurch nicht aufgedeckt werden. Allerdings lässt sich diese Eigenschaft offenbar nicht vermeiden.

5.4. jBPM BPEL-Engine im Test

Aus den in Abschnitt 5.2 genannten Gründen kann die jBPM BPEL-Engine nicht durch die Compliane Testsuite getestet werden. Deshalb kann an dieser Stelle keine Testdurchführung und auch kein Testergebnis angegeben werden.

Dennoch soll das Ant-Skript erläutert werden, das die Tests durchführen würde. Es ist so konzipiert, dass es vermutlich funktioniert, wenn die BPEL-Engine und das Deployment korrekt arbeiten würden. Dies lässt sich allerdings nicht mit genauer Sicherheit sagen, da das Skript nicht getestet werden konnte.

Die BPEL-Prozesse liegen in einer archivierten Form vor. Zudem enthält das Skript Informationen, die für das Deployment benötigt werden.

Das Ant-Skript arbeitet wie folgt: Das Skript startet zuerst die BPEL-Engine. Anschließend deployt es die archivierten BPEL-Prozesse auf der BPEL-Engine. Zum Schluss wird das Deployment der BPEL-Prozesse überprüft, der Server heruntergefahren und eine Report-Datei erstellt.

6. Zusammenfassung

6.1. Kritische Würdigung

In diesem Abschnitt sollen die Probleme dargestellt werden, die im Verlauf der Erstellung dieser Arbeit entstanden sind und die diese Arbeit mit sich gebracht hat.

Zuerst muss man erwähnen, dass BPEL eine sehr junge Sprache ist und aus diesem Grund verhätnismäßig wenig Informationsmaterial und Online-Foren im Vergleich zu zum Beispiel Java existiert.

Diese Tatsache hat die Bearbeitung des Themas dieser Arbeit etwas erschwert und könnte ein Aspekt dafür sein, dass der Fortschritt nach der ersten Hälfte der Bearbeitungszeit nicht meinen Erwartungen entsprach.

Ein weiterer Grund für die leichten Verzögerungen zu Beginn war mein Wunsch, eine möglichst präzise und korrekte theoretische Herleitung des Themas zu Papier zu bringen. Dies führte an einigen Stellen zu einer etwas zu genauen Bearbeitung des Themas.

Da der von mir erstellte Zeitplan einige Puffer enthielt, war es ein leichtes, für die zweite Hälfte einen neuen Zeitplan zu erstellen, der gut eingehalten werden konnte und gegen Ende sogar noch etwas Freiraum bot.

In der Implementierungsphase sind, eventuell bedingt durch den noch nicht fehlerfreien Zustand der Entwicklungstools, einige Fehler aufgetreten, die Zeit gekostet haben und teilweise nur durch Neuinstallation der Tools behoben werden konnten.

Die schwerwiegendsten Probleme lagen dabei bei den BPEL-Engines, die ich installieren und testen wollte. Die erste BPEL-Engine, ActiveBPEL, brachte diverse Probleme bezüglich des Deployments mit sich, da die Tools der kostenpflichtigen ActiveVOS Variante fehlten. Dies endetete damit, dass zur Lösung dieser Probleme offenbar die WSDL-Datei verändert werden musste, was den Grundsätzen der Compliance Testsuite widersprach.

Es wäre nun möglich gewesen, Testfälle zu erstellen und diese laufen zu lassen. Dabei wären alle Testfälle fehlgeschlagen und ActiveBPEL wäre somit laut Compliance Testsuite in keinster Weise mit anderen BPEL-Engines kompatibel gewesen.

Da ich allerdings eine BPEL-Engine auswählen wollte, bei der die Testfälle zumindest teilweise erfolgreich abschließen würden, habe ich mich nach anderen BPEL-Engines umgesehen.

Wie schon in Abschnitt 5.2 beschrieben, gab es diesbezüglich sehr wenig Auswahl. Meine Wahl fiel dann auf die JBPM BPEL-Engine. Diese verursachte allerdings nicht

weniger Probleme. Die Dokumentation der BPEL-Engine war schlecht. An einigen Stellen wurde auf Prozesse und Funktionen verwiesen, die nicht vorhanden waren. Aktuelles Material war im Internet kaum zu finden.

Schlussendlich schlug das Vorhaben, diese BPEL-Engine zu testen ebenfalls fehl, da der Server nicht korrekt aufrufbar war. Einige Seiten ließen sich öffnen, andere Seiten, unter anderem die Seite, die zur Überprüfung des Deployments nötig gewesen wäre, nicht.

Ein weiteres Problem war die Umfrage unter den Benutzern der BPEL-Engine. Es gab so gut wie kein Feedback, wodurch diese Methode zur Gewinnung von Testfällen ausgeschlossen werden musste. Außerdem gab es kaum frei zugängliche Foren, in denen eine Umfrage hätte gestellt werden können.

Probleme bezüglich der BPEL-Engine, die in entsprechenden Foren von mir geäußert wurden, wurden nicht beantwortet. Dies war abzusehen, da bereits Themen mit gleichartigen Probleme seit längerer Zeit existierten, jedoch nie beantwortet wurden.

Active Endpoints bot das entsprechende Forum sogar ausschließlich für zahlende Käufer von ActiveVOS an, obwohl damit geworben wurde, dass die BPEL-Engine ActiveBPEL kostenlos angeboten wird.

Auf Grund dieser Probleme konnte ich leider nicht alle Vorgaben umsetzen, bin aber der Meinung, den wesentlichen Kern dieser Arbeit, nämlich die Herleitung eines Verfahrens zur Entwicklung einer Compliance Testsuite, klar und gut bearbeitet zu haben.

6.2. Ausblick

Im Anschluss an diese Bachelorarbeit ergeben sich einige Möglichkeiten. Der theoretische Grundstein zur Ableitung von Testfällen und damit verbundenen Entwicklung einer Compliance Test Suite ist gelegt.

Nun lassen sich weitere Testfälle erstellen, wodurch die Compliance Testsuite weiter vervollständigt werden kann. Dabei kann versucht werden, einen maximalen Abdeckungsgrad des BPEL-Standards zu erreichen, um so ein gutes Maß für BPEL-Engines zu bekommen.

Dabei ist zu hoffen, dass die Kompatibilität der Engines untereinander steigt, sobald ein Tool existiert, das diese Eigenschaft testen kann.

Desweiteren kann versucht werden, die hier nicht erreichten Punkte abzuhandeln. Es könnten beispielsweise weitere Umfragen durchgeführt werden. Diese müssten allerdings auf einer anderen Ebene stattfinden. Man könnte zum Beispiel Umfragen in Unternehmen durchführen, die BPEL-Engines benutzen.

Der zweite Punkt wäre das erfolgreiche Testen der hier nicht funktionierenden BPEL-Engines in Angriff zu nehmen. Dabei sollte zumindest im Falle von ActiveBPEL umfassend hinterleuchtet werden, wie die Probleme zustande kommen.

Interessant wäre es auch, kostenpflichtige BPEL-Engines zu testen, da man unter

Umständen ein besseres Ergebnis erwarten würde, als bei den kostenlosen. Diesbezüglich wäre der Vergleich einer kostenlosen und einer kostenpflichtigen BPEL-Engine interessant, um zu prüfen, ob der Preis gerechtfertigt ist.

6.3. Fazit

Das Problem, das dieser Bachelorarbeit zu Grunde lag, war die teilweise starke Inkompatibilität der einzelnen BPEL-Engines untereinander. Wünschenswert wäre dabei die Sicherstellung der Interoperabilität der BPEL-Prozesse.

Dies ist insbesondere deswegen wichtig, da der Umstieg der Unternehmen auf eine andere BPEL-Engine im Moment sehr schwierig ist. Im Falle eines solchen Umstiegs müssten in den meisten Fällen sämtliche BPEL-Prozesse an die neue Engine angepasst werden.

Die in dieser Bachelorarbeit zu erstellende Compliance Testsuite hat die Intention dieses Problem zu lösen oder zumindest messbar und somit sichtbar zu machen. Indem eine Compliance Testsuite entwickelt wird, wird den Benutzern der BPEL-Engines ein Werkzeug in die Hand gelegt welches die Wahl einer BPEL-Engine erleichtern soll. Dadurch werden die Entwickler der BPEL-Engines dazu angehalten, ihre Engines genau dem Standard entsprechend zu implementieren, was wiederum die Interoperabilität fördert.

Um dieses Ziel zu erreichen wurden zunächst die theoretischen Grundsteine für eine Compliance Testsuite gelegt. Es wurde hinterleuchtet, was eine Compliance Testsuite enthalten und leisten muss. Anschließend wurde dargestellt, wie Testfälle für eine Testsuite abgeleitet und erstellt werden können. Zum Schluss wurde eine BPEL-Engine durch die Compliance Testsuite getestet, wodurch sich gezeigt hat, was die Testsuite zu leisten im Stande ist.

Die Compliance Testsuite ist in der momentanen Gestalt keinesfalls in der Lage, BPEL-Engines komplett auf ihre Nähe zum BPEL-Standard zu testen. Sie ist allerdings der erste Schritt in diese Richtung und kann, wenn sie entsprechend vervollständigt wurde, ein gutes Maß für die Interoperabilität von BPEL-Prozessen darstellen.

Literaturverzeichnis

[1] Daniel Lübke, An Integrated Approach for Generation in SOA Projects, Verlag Dr. Kovač, Mai 2008

[2] S. Bradner, Key words for use in RFCs to Indicate Requirement Levels, http://www.ietf.org/rfc/rfc2119.txt, letzter Zugriff: 20.07.2009

[3] OASIS, Web Services Business Process Execution Language Version 2.0, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, letzter Zugriff: 10.08.2009

[4] Apache ODE, WS-BPEL 2.0 Specification Compliance, http://ode.apache.org/ws-bpel-20-specification-compliance.html, letzter Zugriff: 10.08.2009

A. Testfälle

Die folgenden Testfälle wurden in Kapitel 4.3 nicht aufgelistet:

Testfall: *fault handler*-Element Anforderung in Scope		Quelle: Standard
Beschreibung:	Ein Scope kann das Attribut *exitOnStandardFault* enthalten. Ist es auf *yes* gesetzt, darf kein *fault handler*-Element benutzt werden, der einen Standardfehler abfängt.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess, der einen Scope beinhaltet, der das *exitOnStandardFault*-Attribut gesetzt hat. Erstelle ein *fault handler*-Element des Scopes.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Das *exitOnStandardFault*-Attribut wird auf *yes* gesetzt. Das *fault handler*-Element wird derart erstellt, dass es einen Standardfehler abfängt.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Das *exitOnStandardFault*-Attribut wird auf *yes* gesetzt. Das *fault handler*-Element wird derart erstellt, dass es keinen Standardfehler abfängt.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.
	Das *exitOnStandardFault*-Attribut wird auf *no* gesetzt. Das *fault handler*-Element wird derart erstellt, dass es einen Standardfehler abfängt.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.
	Das *exitOnStandardFault*-Attribut wird auf *no* gesetzt. Das *fault handler*-Element wird derart erstellt, dass es keinen Standardfehler abfängt.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: *portType*-Attribut bei *invoke*-Aktivität		Quelle: Standard
Beschreibung:	Das *portType*-Attribut der *invoke*-Aktivität ist optional. Wird es angegeben, muss es mit dem Wert des *portType*-Attributs des entsprechenden *partnerLink*-Elements und dem damit zusammenhängenden *role*-Element übereinstimmen.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess mit einer *invoke*-Aktivität und setze ihr *portType*-Attribut.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Das *portType*-Attribut soll dabei einen anderen Wert bekommen, als der Wert des zugehörigen *partnerLink*-Elements und dem damit zusammenhängenden *role*-Element.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Das *portType*-Attribut soll dabei den gleichen Wert bekommen, wie der Wert des zugehörigen *partnerLink*-Elements und dem damit zusammenhängenden *role*-Element.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: *portType*-Attribut bei *onMessage*-Aktivität		Quelle: Standard
Beschreibung:	Das *portType*-Attribut der *onMessage*-Aktivität ist optional. Wird es angegeben, muss es mit dem Wert des *portType*-Attributs des entsprechenden *partnerLink*-Elements und dem damit zusammenhängenden *role*-Element übereinstimmen.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess mit einer *onMessage*-Aktivität und setze ihr *portType*-Attribut.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Das *portType*-Attribut soll dabei einen anderen Wert bekommen, als der Wert des zugehörigen *partnerLink*-Elements und dem damit zusammenhängenden *role*-Element.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Das *portType*-Attribut soll dabei den gleichen Wert bekommen, wie der Wert des zugehörigen *partnerLink*-Elements und dem damit zusammenhängenden *role*-Element.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: Aufruf der *compensate*-Aktivität		Quelle: Standard
Beschreibung:	Die *compensate*-Aktivität darf nur aus einem *fault handler*-Element, einem anderen *compensation handler*-Element oder einem *termination handler*-Element heraus aufgerufen werden.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess mit einer *compensate*-Aktivität und rufe die *compensate*-Aktivität in diesem BPEL-Prozess auf.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Die *compensate*-Aktivität soll dabei nicht aus einem *fault handler*-Element, einem anderen *compensation handler*-Element oder einem *termination handler*-Element heraus aufgerufen werden.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Die *compensate*-Aktivität soll dabei aus einem *fault handler*-Element heraus aufgerufen werden.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.
	Die *compensate*-Aktivität soll dabei aus einem anderen *compensation handler*-Element als dem *fault handler*-Element heraus aufgerufen werden.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.
	Die *compensate*-Aktivität soll dabei aus einem *termination handler*-Element heraus aufgerufen werden.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: Forderung des *initializePartnerRole*-Attributs		Quelle: Standard
Beschreibung:	Das *initializePartnerRole*-Attribut eines *partnerLink*-Elements darf nicht auf *yes* gesetzt werden, wenn das *partnerRole*-Attribut des *partnerLink*-Elements nicht angegeben wurde.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess, in dem ein *partnerLink*-Element definiert wird. Erstelle zudem ein zugehöriges *partnerLinkType*-Element in der entsprechenden WSDL-Datei.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Es soll das *partnerRole*-Attribut nicht angegeben und das *initializePartnerRole*-Attribut auf *yes* gesetzt werden.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Es soll das *partnerRole*-Attribut gesetzt und das *initializePartnerRole*-Attribut mit *yes* belegt werden.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.
	Es soll das *partnerRole*-Attribut gesetzt und das *initializePartnerRole*-Attribut mit *no* belegt werden.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.
	Es soll das *partnerRole*-Attribut nicht angegeben und das *initializePartnerRole*-Attribut auf *no* gesetzt werden.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: Einzigartigkeit des Namens eines *partnerLink*-Elements in einem Scope		Quelle: Standard
Beschreibung:	Ein *partnerLink*-Element kann in einem Scope angegeben werden. Wird es dort angegeben, darf sein Name von keinen anderen *partnerLink*-Elementen des selben Scopes verwendet werden.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess mit einem Scope und erzeuge zwei *partnerLink*-Elemente in diesem Scope. Erstelle zudem zwei zugehörige *partnerLinkType*-Elemente in der entsprechenden WSDL-Datei.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Setze das *name*-Attribut beider *partnerLink*-Elemente auf den selben Wert.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Setze das *name*-Attribut des ersten *partnerLink*-Elements auf einen vom *name*-Attribut des zweiten *partnerLink*-Elements verschiedenen Wert.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: Attributkominbinationen des *propertyAlias*-Elements		Quelle: Standard
Beschreibung:	Ein *propertyAlias*-Element muss entweder die Attribute *messageType* und *part*, das Attribut *type* oder das Attribut *element* definieren.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle eine WSDL-Datei, die ein *propertyAlias*-Element enthält. Zudem muss sie die zu dem *propertyAlias*-Element zugehörige *property*-Definition enthalten.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Setze das *messageType*-Attribut und das *part*-Attribut des *propertyAlias*-Elements.	Die BPEL-Engine muss die WSDL-Datei kompilieren.
	Setze das *type*-Attribut des *propertyAlias*-Elements.	Die BPEL-Engine muss die WSDL-Datei kompilieren.
	Setze das *element*-Attribut des *propertyAlias*-Elements.	Die BPEL-Engine muss die WSDL-Datei kompilieren.
	Setze das *messageType*-Attribut, das *part*-Attribut und das *type*-Attribut des *propertyAlias*-Elements.	Die BPEL-Engine darf die WSDL-Datei nicht kompilieren.
	Setze das *messageType*-Attribut, das *part*-Attribut und das *element*-Attribut des *propertyAlias*-Elements.	Die BPEL-Engine darf die WSDL-Datei nicht kompilieren.
	Setze das *type*-Attribut und das *element*-Attribut des *propertyAlias*-Elements.	Die BPEL-Engine darf die WSDL-Datei nicht kompilieren.
	Setze das *messageType*-Attribut des *propertyAlias*-Elements.	Die BPEL-Engine darf die WSDL-Datei nicht kompilieren.
	Setze das *part*-Attribut des *propertyAlias*-Elements.	Die BPEL-Engine darf die WSDL-Datei nicht kompilieren.
	Setze keines der Attribute.	Die BPEL-Engine darf die WSDL-Datei nicht kompilieren.
	Setze alle Attribute.	Die BPEL-Engine darf die WSDL-Datei nicht kompilieren.

Testfall: Doppeltes Aliasing einer *property*-Definition		Quelle: Standard
Beschreibung:	Es darf keine zwei *propertyAlias*-Elemente geben, die sich auf die selbe *property*-Definition beziehen, deren Wert des *propertyName*-Attributs gleich ist und die den gleichen WS-BPEL Variablentyp angeben.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle eine WSDL-Datei, die zwei *propertyAlias*-Elemente enthält. Zudem muss sie eine *property*-Definition enthalten. Das *name*-Attribut der *property*-Definition und die beiden *propertyName*-Attribute der *propertyAlias*-Elemente besitzen den selben beliebig gewählten, gültigen, festen Wert.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Setze die beiden *messageType*-Attribute auf den selben beliebig gewählten, gültigen, festen Wert und die beiden *part*-Attribute auf den selben beliebig gewählten, gültigen, festen Wert.	Die BPEL-Engine darf die WSDL-Datei nicht kompilieren.
	Setze die beiden *type*-Attribute auf den selben beliebig gewählten, gültigen, festen Wert.	Die BPEL-Engine darf die WSDL-Datei nicht kompilieren.
	Setze die beiden *element*-Attribute auf den selben beliebig gewählten, gültigen, festen Wert.	Die BPEL-Engine darf die WSDL-Datei nicht kompilieren.
	Setze die beiden *messageType*-Attribute auf einen unterschiedlichen Wert und die beiden *part*-Attribute auf einen unterschiedlichen Wert.	Die BPEL-Engine muss die WSDL-Datei kompilieren.
	Setze die beiden *type*-Attribute auf einen unterschiedlichen Wert.	Die BPEL-Engine muss die WSDL-Datei kompilieren.
	Setze die beiden *element*-Attribute auf einen unterschiedlichen Wert.	Die BPEL-Engine muss die WSDL-Datei kompilieren.

Testfall: Struktur der *invoke*-Aktivität bei fehlenden *part*-Elementen der *message*-Definition		Quelle: Standard
Beschreibung:	Eine *invoke*-Aktivität ruft andere Webservices auf, indem sie ihnen Nachrichten schickt. Wenn die zugehörige *message*-Definition einer solchen Nachricht keine *part*-Elemente enthält, müssen die *toParts*- und *fromParts*- Elemente der *invoke*-Aktivität weggelassen werden.	
Voraussetzungen:	Standard-Voraussetzung 2	
Durchführung:	Erstelle zwei BPEL-Prozesse. Einer der beiden Prozesse soll eine *invoke*-Aktivität enthalten. Erstelle des weiteren eine entsprechende *message*-Definition. Die *message*-Definition soll dabei keine *part*-Elemente enthalten.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Die *invoke*-Aktivität soll mindestens ein gültiges *toParts*-Element enthalten.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Die *invoke*-Aktivität soll mindestens ein gültiges *fromParts*-Element enthalten.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Die *invoke*-Aktivität soll weder ein *toParts*-Element, noch ein *fromParts*-Element beinhalten.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: Auswirkung vorhandener *toPart*-Elemente auf das *inputVariable*-Attribut in der *invoke*-Aktivität		Quelle: Standard
Beschreibung:	In einer *invoke*-Aktivität darf das *inputVariable*-Attribut nicht benutzt werden, wenn *toPart*-Elemente vorhanden sind.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess, der eine *invoke*-Aktivität enthält, in der *toPart*-Elemente vorhanden sind.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Das *inputVariable*-Attribut der *invoke*-Aktivität soll gesetzt sein.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Das *inputVariable*-Attribut der *invoke*-Aktivität soll nicht gesetzt sein.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

Testfall: Auswirkung vorhandener *fromParts*-Elemente auf das *outputVariable*-Attribut in der *invoke*-Aktivität		Quelle: Standard
Beschreibung:	In einer *invoke*-Aktivität darf das *outputVariable*-Attribut nicht benutzt werden, wenn *fromParts*-Elemente vorhanden sind.	
Voraussetzungen:	Standard-Voraussetzung 1	
Durchführung:	Erstelle einen BPEL-Prozess, der eine *invoke*-Aktivität enthält, in der *fromParts*-Elemente vorhanden sind.	
Eingaben:	Eingabe	Erwartetes Ergebnis
	Das *outputVariable*-Attribut der *invoke*-Aktivität soll gesetzt sein.	Die BPEL-Engine darf den BPEL-Prozess nicht kompilieren.
	Das *outputVariable*-Attribut der *invoke*-Aktivität soll nicht gesetzt sein.	Die BPEL-Engine muss den BPEL-Prozess kompilieren.

B. Standardfehler

Die folgenden Fehler werden als Standardfehler bezeichnet:

- ambiguousReceive
- completionConditionFailure
- conflictingReceive
- conflictingRequest
- correlationViolation
- invalidBranchCondition
- invalidExpressionValue
- invalidVariables
- joinFailure
- mismatchedAssignmentFailure
- missingReply
- missingRequest
- scopeInitializationFailure
- selectionFailure
- subLanguageExecutionFault
- uninitializedPartnerRole
- uninitializedVariable
- unsupportedReference
- xsltInvalidSource
- xsltStylesheetNotFound

C. Abkürzungen für Testfälle

Die folgenden Abkürzungen werden für die entsprechenden Testfälle gewählt:

StrukturReceive Struktur der receive-Aktivität bei fehlenden part-Elementen der message-Definition

StrukturReply Struktur der reply-Aktivität bei fehlenden part-Elementen der message-Definition

FromPartReceive Auswirkung der fromPart-Elemente auf das variable-Attribut einer receive-Aktivität

ToPartReply Auswirkung der toPart-Elemente auf das variable-Attribut einer reply-Aktivität

PortTypeReceive portType-Attribut bei receive-Aktivität

PortTypeReply portType-Attribut bei reply-Aktivität

SyncProzess Synchrone und asynchrone BPEL-Prozesse

D. Übersicht der Apache ODE Testergebnisse

Test	Ergebnis
Auswirkung der fromPart-Elemente auf das variable-Attribut einer receive-Aktivität (kompiliert nicht)	failure
Auswirkung der fromPart-Elemente auf das variable-Attribut einer receive-Aktivität (kompiliert)	failure
Auswirkung der toPart-Elemente auf das variable-Attribut einer reply-Aktivität (kompiliert nicht)	failure
Auswirkung der toPart-Elemente auf das variable-Attribut einer reply-Aktivität (kompiliert)	failure
portType-Attribut bei receive-Aktivität (kompiliert nicht)	success
portType-Attribut bei receive-Aktivität (kompiliert)	success
portType-Attribut bei reply-Aktivität (kompiliert nicht)	success
portType-Attribut bei reply-Aktivität (kompiliert)	success
Struktur der receive-Aktivität bei fehlenden part-Elementen der message-Definition (kompiliert nicht)	success
Struktur der receive-Aktivität bei fehlenden part-Elementen der message-Definition (kompiliert)	success
Struktur der reply-Aktivität bei fehlenden part-Elementen der message-Definition (kompiliert nicht)	success
Struktur der reply-Aktivität bei fehlenden part-Elementen der message-Definition (kompiliert)	success
Synchrone und asynchrone BPEL-Prozesse (asynchron)	failure
Synchrone und asynchrone BPEL-Prozesse (synchron)	success

E. Compact Disk

Folgendes Material befindet sich auf der der Bachelorarbeit beigelegten CD:

- Diese Bachelorarbeit
- Die implementierten BPEL-Prozesse
- Das Apache ANT-Skript

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 20.08.2009

_____

Stephan Kiesling