

Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Bachelorarbeit Nr. 153

Modellierung von REST Service Kompositionen

Volha Kalach

Studiengang: Softwaretechnik

Prüfer/in: Prof. Dr. Frank Leymann

Betreuer/in: Dipl.-Inf. Florian Haupt
Dipl.-Inf. Karolina Vukojevic-Haupt

Beginn am: 15.05.2014

Beendet am: 14.11.2014

CR-Nummer: H.4.1, H.5.2

Kurzfassung

In Vorarbeiten wurde eine Erweiterung der BPEL-Sprache (BPEL4REST) definiert und implementiert, die eine Orchestrierung von REST Services mit BPEL ermöglicht. Die Erweiterung bietet jedoch keine grafische Benutzerschnittstelle für die Modellierung von BPEL-Prozessen. Dies erschwert die Anwendung von BPEL4REST.

In dieser Arbeit wird eine Erweiterung des SimTech BPEL Designers konzipiert und prototypisch implementiert. Diese Erweiterung bietet eine grafische Unterstützung für die Modellierung von REST Service Kompositionen mit BPEL an.

Inhaltsverzeichnis

1	Einleitung	9
1.1	Ziel der Bachelorarbeit	9
1.2	Gliederung	10
2	Grundlagen	11
2.1	REST	11
2.2	BPEL und BPEL Designer	13
2.2.1	BPEL Designer	14
2.2.2	SimTech BPEL Designer	16
3	BPEL4REST	19
3.1	Motivation und Überblick	19
3.2	Aufbau der BPEL4REST Extension Activities	19
3.2.1	Die Struktur des <put>-Elements	20
3.2.2	Die Struktur des <context>-Elements	21
3.2.3	Die Struktur des <requestParameters>-Elements	22
3.2.4	Die Struktur des <responseParameters>-Elements	23
4	Entwurf	25
4.1	Allgemeine Konzepte	25
4.2	UI Prototyp	27
4.2.1	Reiter „Details“	28
4.2.2	Reiter „Context“	28
4.2.3	Reiter „Request Parameters“	29
4.2.4	Reiter „Response Parameters“	31
4.3	Validierung	32
5	Implementierung	33
5.1	Entwicklungs- und Testumgebung	33
5.1.1	Eclipse	33
5.1.2	SimTech BPEL Designer	33
5.2	Einbettung der RESTModelingExtension in den BPEL Designer	34
5.3	Architektur	36
5.3.1	Übersicht der Architektur	36
5.3.2	Model-Komponente	37
5.3.3	View-Komponente	40

5.3.4	Controller-Komponente	42
5.3.5	Validierung	43
5.4	Einschränkungen	45
6	Zusammenfassung und Ausblick	47
6.1	Ausblick	47
	Literaturverzeichnis.....	49

Abbildungsverzeichnis

Abbildung 2.1: Ableitung der REST [2].....	11
Abbildung 2.2: Eclipse BPEL Designer	15
Abbildung 4.1: Eine mögliche Darstellung einer REST-Aktivität im BPEL-Prozess	26
Abbildung 4.2: Belegung des Reiters "Details"	27
Abbildung 4.3: Belegung des Reiters "Context"	28
Abbildung 4.4: Belegung des Reiters "Request Parameters"	29
Abbildung 4.5: Dialogfenster für das Hinzufügen einer neuen Accept Entity	30
Abbildung 4.6: Belegung des Reiters "Response Parameters"	31
Abbildung 5.1: Beziehungen zwischen den RESTModelingExtension-Plug-ins und Plug-ins vom BPEL Designer	35
Abbildung 5.2: Übersicht der Architektur	37
Abbildung 5.3: Ausschnitt aus dem Klassendiagramm des EMF-Modells	38
Abbildung 5.4: Transformationen zwischen den XML-Elementen und Klassen des EMF-Modells	40
Abbildung 5.5: Vererbungshierarchie der PropertySection-Klassen	41
Abbildung 5.6: Zusammenarbeit MVC	43
Abbildung 5.7: Ausschnitt aus dem Vererbungsdiagramm der Validator-Klassen	44

Verzeichnis der Listings

Listing 3.1: Struktur des XML-Elements <put>.....	20
Listing 3.2: Struktur des <context>-Elements	21
Listing 3.3: Struktur des XML-Elementes <requestParameters>	22
Listing 3.4: Struktur des <responseParameters>-Elementes	23

1 Einleitung

Seit Langem sind Computer ein Bestandteil jedes Unternehmens unabhängig vom Bereich, in welchem die Unternehmen tätig sind. Und während früher Desktopanwendungen die Hauptrolle spielten, gewinnen jetzt Web-Anwendungen und Web Services immer mehr an Bedeutung. Auch wird großer Wert auf die Systeme mit einer losen Kopplung und Wiederverwendbarkeit von Komponenten gelegt.

Service Oriented Computing (SOC) bezeichnet die Verwendung von Services als grundlegende Komponente für die Entwicklung von Anwendungen. Für die Erstellung einer neuen Anwendung werden die vorhandenen Services miteinander kombiniert; dabei können aus mehreren einfachen Services viel kompliziertere Anwendungen entstehen, die wiederum als Services angeboten werden können. Dieser Prozess wird als *Service Komposition* bezeichnet [1].

Für die Entwickler, die nach SOC Prinzipien arbeiten, ist es wichtig, eine passende Vorgehensweise zur Zusammensetzung des Services zu wählen. Als eine der Möglichkeiten bietet sich die Verwendung von BPEL, als eine Sprache für die Orchestrierung WSDL-basierter Web Services, siehe Kapitel 2.2.

Allerdings bieten nicht alle Services eine WSDL-Schnittstelle an. Der im Jahr 2000 eingeführte Architekturstil REST schlägt ein alternatives Verfahren zur Erstellung der Services vor, siehe Kapitel 2.1. Die Services, die nach den Prinzipien des REST-Stils entwickelt wurden, werden im Folgenden als REST Services bezeichnet.

Um trotz der fehlenden WSDL-Schnittstelle die REST Services mit BPEL aufrufen zu können, wurde im Rahmen einer Vorarbeit eine Erweiterung der BPEL definiert, die eine Orchestrierung von REST Services ermöglicht, siehe Kapitel 3.

Ein wichtiger Aspekt bei Service Kompositionen ist die Unterstützung der Modellierer mit hilfreichen Werkzeugen, die unter anderem grafische Mittel für die Modellierung anbieten. Zwar wird der Eclipse BPEL Designer (siehe Kapitel 2.2.1) als ein Open Source Editor für BPEL verwendet, doch dieser Editor bietet keine Unterstützung zur grafischen Modellierung der Elemente, die in der oben erwähnten Erweiterung eingeführt wurden. Aus diesem Grund entstand der Bedarf, in dieser Arbeit die Möglichkeiten zur Erweiterung des BPEL Designers zu untersuchen.

1.1 Ziel der Bachelorarbeit

Ziele dieser Arbeit sind:

- Untersuchung der Möglichkeiten zur Unterstützung eines Modellierer bei der Modellierung von REST Service Kompositionen mit BPEL,
- Analyse der BPEL4REST-Erweiterung in Bezug auf die Modellierungsphase,
- Erstellung eines Konzepts für grafische Unterstützung der Modellierungsaufgaben,

- prototypische Implementierung dieses Konzepts einschließlich der Integration in SimTech BPEL Designer.

1.2 Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen: In Kapitel 2 werden die grundlegenden Begriffe wie REST und BPEL erklärt, die für das Verständnis der Arbeit Voraussetzung sind. Außerdem werden die grafischen Editoren für BPEL angesprochen.

Kapitel 3 – BPEL4REST: Das dritte Kapitel beschreibt eine bestehende Erweiterung der BPEL, die Möglichkeiten anbietet, REST Services mit BPEL zu orchestrieren.

Kapitel 4 – Entwurf: In diesem Kapitel werden Möglichkeiten des grafischen Designs für BPEL4REST analysiert und ein UI Prototyp vorgeschlagen.

Kapitel 5 – Implementierung: Dieses Kapitel beschäftigt sich mit den Implementierungsaspekten der Erweiterung für BPEL Designer.

Kapitel 6 – Zusammenfassung: Das Kapitel 6 fasst die Ergebnisse der Arbeit zusammen und gibt einen Ausblick.

2 Grundlagen

In diesem Kapitel werden die grundlegenden Begriffe beschrieben, die in dieser Arbeit verwendet werden und deren Kenntnis zum allgemeinen Verständnis der Arbeit dient. Zuerst wird REST vorgestellt, danach die BPEL-Spezifikation besprochen.

2.1 REST

Quelle: [2]

Representational State Transfer (REST) ist ein Architekturstil für verteilte Hypermedia Systeme, der im Jahr 2000 in der Dissertation von Roy T. Fielding [2] eingeführt und ausgearbeitet wurde. Im Rahmen seiner Arbeit analysiert Fielding existierende Architekturstile und stellt Anforderungen an eine Web-Architektur, wie Einfachheit, Erweiterbarkeit, verteilte Hypermedia, Skalierbarkeit und unabhängiges Deployment. Um REST zu definieren, fügt Fielding iterativ neue Einschränkungen zu einem initialen Architekturstil hinzu, dabei wird ein hybrider Stil abgeleitet, der die gewünschten Eigenschaften einer Web-Architektur besser reflektiert (siehe Abbildung 2.1).

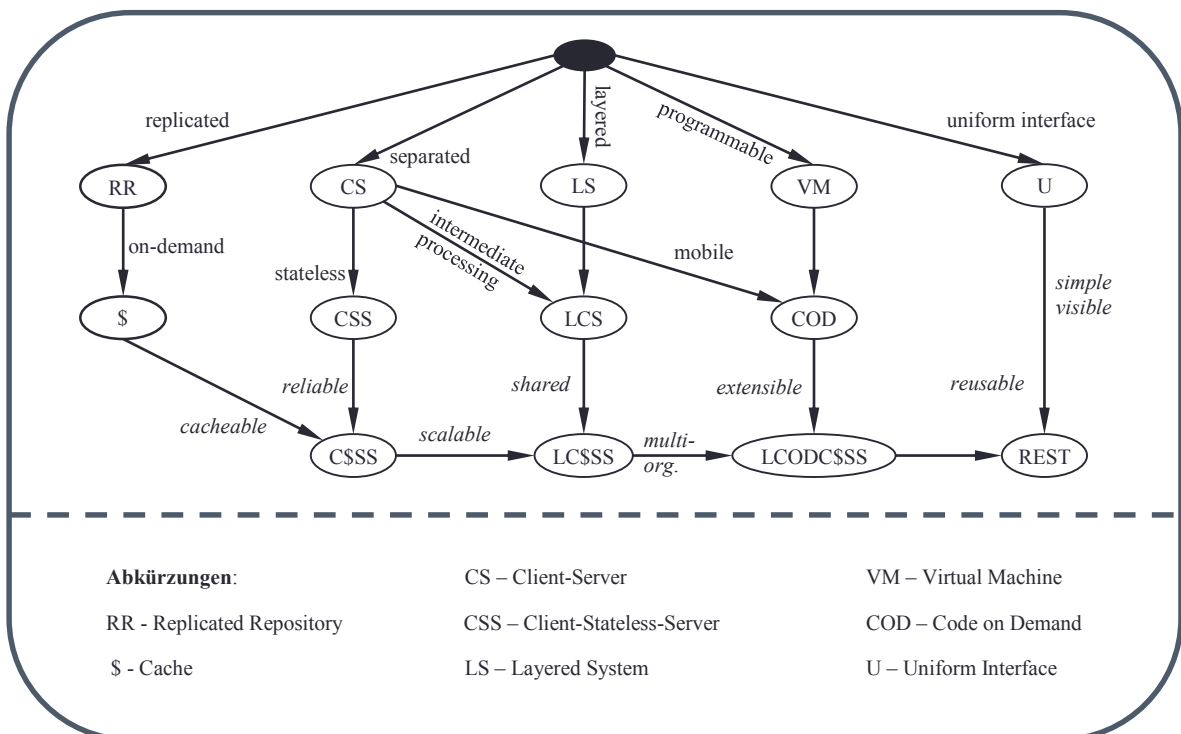


Abbildung 2.1: Ableitung der REST [2]

Angefangen mit einem „Null Style“, der keine Einschränkungen enthält, werden als Erstes die Einschränkungen des *Client-Server*-Architekturstils hinzugefügt. Die Trennung der Zuständigkeiten erhöht die Portabilität der Benutzerschnittstelle und ihre Skalierbarkeit und lässt außerdem die unabhängige Entwicklung der Komponenten zu.

Als Nächstes wird die Einschränkung hinzugefügt, dass die Kommunikation „*stateless*“ sein soll. Dies vereinfacht die Implementierung des Servers, da keine Daten zwischen Abfragen gespeichert werden sollen, und führt zur Erhöhung der Skalierbarkeit, Sichtbarkeit und Zuverlässigkeit. Hier entsteht jedoch ein Nachteil, nämlich dass die gleichen Informationen mehrmals innerhalb einer Folge von Abfragen gesendet werden. Dies führt zum Sinken der Netzwerkperformanz.

Um diesen Nachteil auszugleichen und die Netzwerkeffizienz zu erhöhen, wird eine weitere Einschränkung eingeführt: Die Daten in einer Antwort auf eine Anfrage sollen als „*cacheable*“ oder „*non-cacheable*“ markiert werden. Die *cacheable* Daten können später als Antwort für eine äquivalente Anfrage wiederverwendet werden.

Identifikation der Ressourcen, Manipulation der Ressourcen mittels Repräsentation, selbstbeschreibende Nachrichten und Hypermedia as the engine of application state (HATEOAS) sind die vier Einschränkungen an eine Schnittstelle, deren Einbringen zu einer *einheitlichen Schnittstelle* zwischen Komponenten führt.

Die *Ressource* ist das wichtigste Merkmal von REST. Alle Informationen, die benannt werden können, sind Ressourcen. Um Ressourcen zu identifizieren, werden Ressourcenbezeichner verwendet. Die Ressourcen können wiederum mehrere verschiedene *Repräsentationen* haben. Dabei besteht eine Repräsentation einer Ressource aus einer Bytesequenz sowie aus Metadaten, die diese Bytesequenz beschreiben. Für die Beschreibung des Datenformats einer Repräsentation werden MIME Media Typen [3] benutzt. Abhängig von existierenden Repräsentationen einer Ressource und Parametern der Anfrage wird mittels Content Negotiation die am meisten passende Repräsentation für eine Antwortnachricht ausgewählt. Außerdem werden die Ressourcen mittels ihrer Repräsentationen manipuliert, wenn die in einer Anfrage enthaltene Repräsentation zur Erstellung oder Änderung einer Ressource dient. Die Verwendung der selbstbeschreibenden Nachrichten bedeutet, dass jede Anfragenachricht alle nötigen Informationen enthalten soll, um vom Server verstanden zu werden. Und letztendlich HATEOAS wird dadurch dargestellt, dass die Repräsentationen von Ressourcen Links auf andere Ressourcen enthalten können.

Als Folge der Nutzung der einheitlichen Schnittstelle wird, laut Fielding, die gesamte Systemarchitektur vereinfacht, die Sichtbarkeit der Interaktionen verbessert und die Implementierung von Service entkoppelt. Die einheitliche Schnittstelle hat allerdings im Vergleich zu einer auf die Anwendung angepassten Schnittstelle eine geringere Effizienz.

Das *Layered System*, das als nächste Einschränkung von Fielding hinzugefügt wurde, schreibt vor, dass das gesamte System aus mehreren Schichten bestehen soll und seine Komponenten nur mit unmittelbaren Nachbarn kommunizieren können. Besonders in Kombination mit früheren Einschränkungen wie Caching und der einheitlichen Schnittstelle kann das Layered System zu einer wesentlichen Erhöhung der Performanz führen, denn Antwortnachrichten können eine „*cached*“ Response enthalten, die initial für einen anderen Client erstellt wurde.

Als letztes führt Fielding die Einschränkungen des „*Code-On-Demand*“-Stils ein, die zu einer Erweiterbarkeit der Funktionalität dienen. Die Erweiterungen werden durch das Herunterladen und Ausführen des Codes in Applet- oder Skript-Form ermöglicht. Diese letzte Einschränkung betrachtet Fielding als optional.

Im Weiteren evaluiert Fielding den definierten REST-Stil und beschreibt seine Erfahrungen während dessen Anwendung am Hypertext Transfer Protocol (HTTP) [4] und dem Uniform Resource Identifier (URI) [5]. Hier kann man bemerken, dass die Verwendung des HTTP Protokolls zusammen mit URI und MIME schon die meisten REST-Einschränkungen impliziert. Dem Entwickler wird überlassen den Server „stateless“ zu implementieren und dafür zu sorgen, dass die HATEOAS Einschränkung erfüllt ist [6].

Mittlerweile ist REST weit verbreitet; verschiedene Unternehmen wie Google, Amazon usw. bieten REST-Schnittstellen für ihre Systeme an. Die Systeme, die im REST-Stil gebaut sind, unterscheiden sich in vielen Aspekten von denen, die SOAP [7] implementieren. Im Gegensatz zu REST hat jede SOAP-Anwendung eine eigene Schnittstelle, alle SOAP-Nachrichten verwenden bei der Nutzung HTTP-Protokolle die HTTP POST Methode, die an eine gegebene Adresse gesendet wird.

Zusammenfassend lässt sich feststellen, dass die Entwicklung von Systemen nach REST-Architekturstil verschiedene Vorteile mit sich bringt. Unter anderem sind die lose Kopplung, Interoperabilität, Wiederverwendung, Performance und Skalierbarkeit die Eigenschaften, die REST Services besitzen [8].

2.2 BPEL und BPEL Designer

Web Service Business Process Execution Language (BPEL) ist eine XML basierte Sprache, die Geschäftsprozesse modellieren lässt. Im Unterschied zu anderen Sprachen und Notationen, die zur Modellierung der Geschäftsprozesse dienen, werden Geschäftsprozesse in BPEL ausschließlich mit XML beschrieben, keine grafische Darstellung ist hierzu spezifiziert. Die erste Spezifikation von BPEL wurde im Jahr 2002 von OASIS veröffentlicht. Derzeit ist die Version WS-BPEL 2.0 aktuell [9], dabei kündigte das Technische Komitee an, dass die Arbeit an der Spezifikation abgeschlossen sei.

BPEL unterscheidet zwei Arten von Geschäftsprozessen: die abstrakten und die ausführbaren Prozesse. Die abstrakten Prozesse werden nur teilweise spezifiziert und sind nicht für eine Ausführung vorbestimmt. Allerdings bestehen keine Beschränkungen auf BPEL-Konstruktionen, die in abstrakten Prozessen verwendet werden dürfen. Außerdem bietet diese Art von Prozessen spezielle Mechanismen zum Verstecken der operationalen Details an. Die abstrakten Prozesse können für die Erstellung eines Templates verwendet werden.

Die ausführbaren BPEL-Prozesse beschreiben den Verlauf des Geschäftsprozesses basierend auf Interaktionen zwischen dem Prozess und seinen Partnern. Sowohl diese Interaktion, als auch die Interaktion mit dem BPEL-Prozess selbst verläuft mittels einer Web-Service-Schnittstelle, die mit WSDL 1.1 beschrieben ist. Außer WSDL 1.1 verwendet BPEL für die Definition der Geschäftsprozesse auch weitere XML basierte Spezifikationen. XML Schema 1.0 wird benutzt, um das Datenmodell des BPEL-Prozesses zu beschreiben. XPath 1.0 und XSLT 1.0 werden für Datenmanipulationen verwendet.

Ein BPEL-Prozess hat eine klare Struktur. Sämtliche Interaktionen eines Geschäftsprozesses werden als Aktivitäten dargestellt. Dabei kann man verschiedene Arten der Aktivitäten auszeichnen. Einige davon werden in den folgenden Abschnitten kurz erläutert.

Die Aktivitäten, die für eine Interaktion mit Prozesspartner vorbestimmt sind, sind `<invoke>`, `<receive>` und `<reply>`. Alle drei haben `partnerLink` und `operation` als verpflichtende Attribute, die benutzt werden, um den Interaktionspartner zu bestimmen. `<receive>` wartet auf passende Nachricht. `<reply>` sendet eine Nachricht zurück als Antwort auf eine Eingegangene. `<invoke>` ruft eine eingegebene Operation des Partners auf und kann eventuell auf eine Antwortnachricht warten.

Zudem kann man eine Gruppe der Aktivitäten auszeichnen, die für die Kontrollflusssteuerung verantwortlich sind. Diese Aktivitäten beinhalten weitere Aktivitäten als Kind-Elemente, deren Ausführungsreihenfolge hängt von der Aktivitätsart und eventuell weiteren zusätzlichen Bedingungen ab. Alle Kind-Aktivitäten der `<sequence>` werden nacheinander ausgeführt. `<if>` wertet die Bedingung aus und lässt nur eine der Alternativen ausführen. `<while>`, `<repeatUntil>` und `<forEach>` können die Kind-Aktivität mehrmals wiederholt ausführen. `<flow>` lässt die parallele Ausführung von Kind-Aktivitäten zu. `<wait>` und `<pick>` warten bis ein bestimmtes Ereignis vorkommt.

Als weiteres kann man Aktivitäten, die zur Fehlerbehandlung dienen, ansprechen. Dazu gehören `<throw>` und `<rethrow>`, die eine Fault generieren können, `<compensate>` und `<compensateScope>`, die eine Ausführung der kompensierenden Aktivitäten starten, und `<exit>`, die zur sofortigen Beendigung des BPEL-Prozesses führt.

Alle in dieser Aktivität verwendeten Elemente müssen deklariert werden. So enthält ein BPEL-Prozess Deklarationen der `PartnerLinks`, `Variablen`, eventuellen `faultHandlers`, `eventHandlers` und anderen Elementen. Die Deklaration kann sowohl global für den ganzen BPEL-Prozess erfolgen, als auch lokal für einen bestimmten `<scope>`.

Um einen BPEL-Prozess auszuführen, wird er in eine BPEL-Engine deployed. Auf dem Markt sind sowohl kommerzielle als auch Open-Source BPEL-Engine vorhanden. Apache – einer der Open Source Anbieter – bietet „*Orchestration Director Engine*“ (Apache ODE) an. Die Software interagiert mit Web Services, sendet und empfängt Nachrichten, bearbeitet Daten und führt Fehlerbehandlung durch, wie es im BPEL-Prozess definiert wurde [10].

2.2.1 BPEL Designer

Wie oben erwähnt, schreibt die BPEL-Spezifikation keine grafische Darstellung der Elemente des BPEL-Prozesses vor. Allerdings wäre es für Entwickler viel verständlicher und einfacher mit grafischen Elementen zu arbeiten als einen XML-Code zu schreiben. Manche kommerziellen Anbieter der BPEL-Engine haben in ihrem Angebot auch Modellierungswerkzeuge für BPEL-Prozesse. Zum Beispiel beinhaltet Oracle BPEL Process Manager einen BPEL Process Designer, ein Plug-in für JDeveloper [11]. Ein Vertreter der Open Source Modellierungswerkzeuge für BPEL ist der *Eclipse BPEL Designer*. Das ist ein GEF basierter Eclipse-Editor. Er bietet Unterstützung für Entwicklung, Editieren, Deployen, Testen und Debug der WS-BPEL 2.0 Prozesse [12].

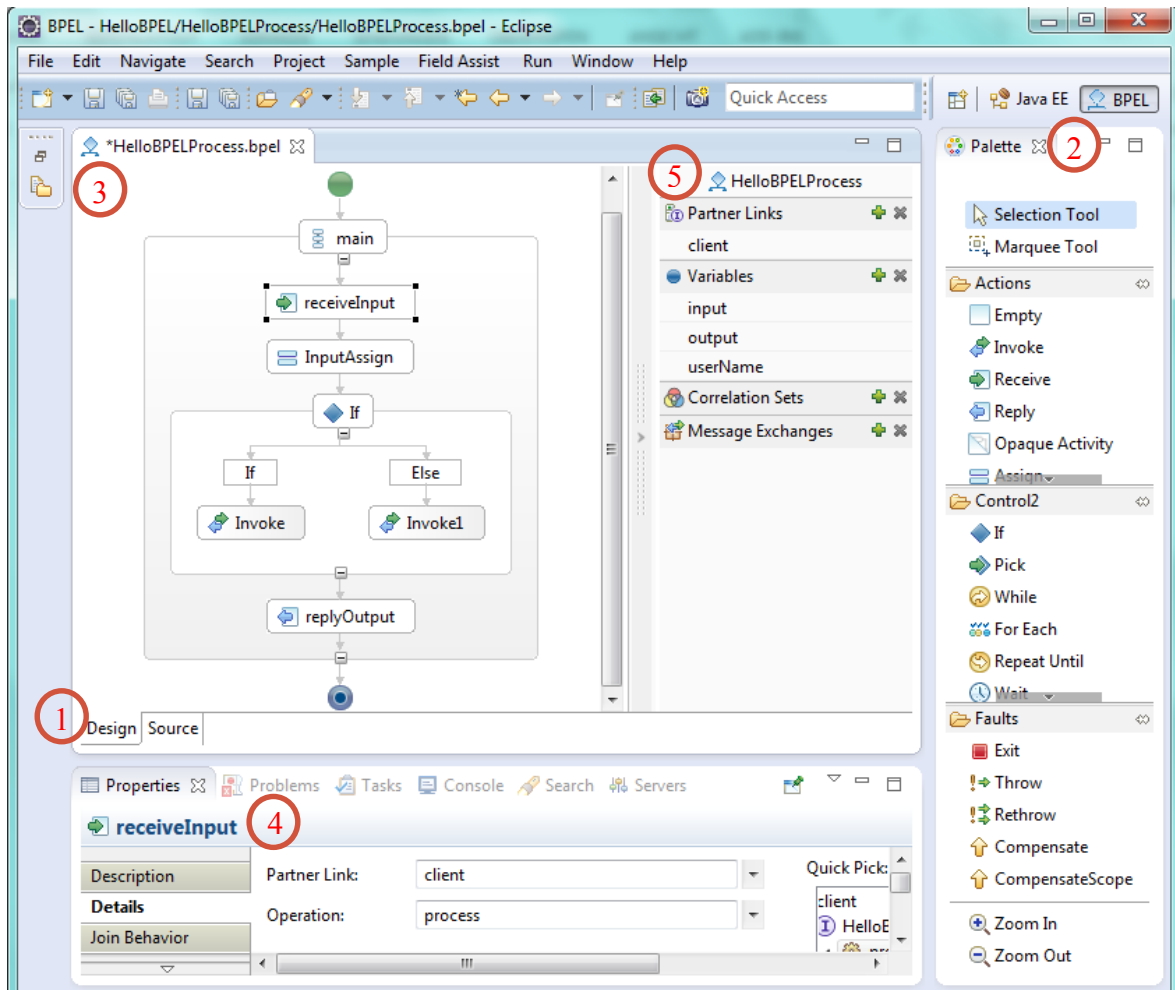


Abbildung 2.2: Eclipse BPEL Designer

Auf der Abbildung 2.2 ist der Eclipse BPEL Designer zu sehen. Die Zwei Ansichten des BPEL Designers (1) – Design und Source – bieten den Nutzern Flexibilität, ermöglichen volle Kontrolle über die Erstellung des BPEL-Prozesses und lassen dabei eine gute Übersicht über den ganzen BPEL-Prozess zu. Die Palette (2) enthält alle in der BPEL-Spezifikation beschriebenen Aktivitäten. Die Aktivitäten lassen sich mit der Drag-and-drop Technik in den BPEL-Prozess (3) einfügen. Die Attribute einer Aktivität lassen sich in der Ansicht „Properties“ (4) editieren.

Die zusätzliche Leiste (5) der Ansicht „Design“ dient zur Verwaltung der Deklarationen, sowohl des gesamten BPEL-Prozesses als auch jedes einzelnen Scopes. Hier werden neue Variablen, Partner-Links usw. angelegt und gelöscht. Außerdem bietet BPEL Designer zahlreiche Wizards und Dialogs für das Erstellen bzw. die Auswahl von Elementen des BPEL-Prozesses.

Alle Interaktionen, die der Benutzer in der Ansicht „Design“ durchführt, beeinflussen den Inhalt der BPEL-Datei, die in der Ansicht „Source“ angezeigt werden kann. Dies gilt auch für die andere Richtung: alle manuellen mit der BPEL-Spezifikation konformen Änderungen der BPEL-Datei werden in der „Design“-Ansicht sichtbar.

Außerdem bietet der Eclipse BPEL Designer Möglichkeiten, den entstandenen Quellcode der BPEL-Datei zu validieren. Die Validierung wird nach dem Speichern der Datei automatisch durchgeführt. Die gefundenen Fehler werden sowohl in der „Source“- als auch in der „Design“-Ansicht markiert.

2.2.2 SimTech BPEL Designer

Anwendungsgebiete des BPEL beinhalten nicht nur die Modellierung von Geschäftsprozessen, sondern auch die Erstellung wissenschaftlicher Workflows, die die Durchführung von Simulationen steuern. Hier entstehen jedoch neue Anforderungen an die Ausführung des BPEL-Prozesses. Ein und dieselbe wissenschaftliche Simulation wird mit verschiedenen Eingabeparametern durchgeführt. Die Ausführung kann mehrere Stunden dauern. Zudem entsteht Bedarf die Simulation zu unterbrechen und die Zwischenwerte anzuschauen. Schließlich sollen Wissenschaftler mit dem Deployment des BPEL-Prozesses in BPEL-Engine nicht belastet werden.

Das „*SimTech Scientific Workflow Management System*“ (SimTech SWfMS) berücksichtigt oben genannte Anforderungen und stellt damit ein benutzerfreundliches Werkzeug zur Modellierung, Ausführung und Debuggen wissenschaftlicher Workflows vor [13]. SimTech SWfMS besteht aus mehreren Teilen und beinhaltet unter anderem den SimTech BPEL Designer, der basierend auf dem Eclipse BPEL Designer entwickelt wurde, und eine umgebaute Version der ODE, die ODE-PGF genannt wird. Als wesentliche Erweiterungen kann man die folgenden Funktionalitäten von SimTech SWfMS bemerken:

- Das Deployment und die Ausführung von Prozessmodellen per Knopfdruck aus dem Eclipse BPEL Designer.
- Monitoring der Prozessinstanzen, welches durch Einfärbung der Aktivitäten im BPEL Designer die aktuellen Zustände der Aktivitäten während der Ausführung der Prozessinstanz anzeigt.
- Das Anzeigen und Ändern der Werte von Variablen und Partner-Links, während der Ausführung einer Prozessinstanz.
- Parametrisierung der Variablen, die eine Durchführung von Parameterstudien ermöglicht, indem mehrere Prozessinstanzen mit verschiedenen Variablenwerten gleichzeitig gestartet werden.
- Versionierung und Metadatenverwaltung.
- Breakpoints, um die Ausführung der Prozessinstanz zu unterbrechen.
- Beeinflussung des Ausführungsprozesses und Änderungen des Kontrollflusses durch die Benutzeroberfläche.

Ein weiteres Bedürfnis bei der Erstellung einer wissenschaftlichen Simulation ist, die REST Services in den Prozess einzubinden. Dabei ist für die Wissenschaftler wichtig, dass die Aufrufe von REST Services, ähnlich mit den Aufrufen WSDL-basierter Services, im SimTech BPEL Designer grafisch modelliert werden, damit die volle Funktionalität, die der SimTech BPEL Designer anbietet, genutzt werden kann.

Die BPEL-Spezifikation bietet keine Vorschriften in diesem Bereich, da BPEL sich nur an den WSDL-basierten Services orientiert. Eine bestehende Lösungsmöglichkeit wird im Kapitel 3 erläutert.

3 BPEL4REST

In diesem Kapitel wird eine Erweiterung für BPEL vorgestellt, welche neue Aktivitätstypen für die Orchestrierung von REST Services einführt. Diese Erweiterung wird im Folgenden *BPEL4REST* genannt.

3.1 Motivation und Überblick

BPEL4REST wurde im Rahmen der Bachelorarbeit von Markus Fischer [14] konzipiert und implementiert. Anhand durchgeführter Analysen stellt Fischer fest, dass die bestehenden Ansätze zur Orchestrierung von REST Services mit BPEL einige Mängel aufweisen. So erfüllen diese Ansätze nicht alle drei folgenden Anforderungen vollständig, die für eine einwandfreie Nutzung der REST Services erforderlich sind:

- Der URI von HTTP-Anfragen muss zur Laufzeit eines BPEL-Prozesses manipulierbar sein.
- Die Content-Negotiation muss für alle REST-Aufrufe in vollem Umfang unterstützt werden.
- HTTP-Header müssen für alle REST-Aufrufe eines BPEL-Prozesses manipulierbar sein.

Unter Verwendung von WSDL 1.1 ist es unmöglich auf den HTTP-Header zuzugreifen. Apache ODE WSDL 1.1 Extension for REST führt eine Unterstützung des HTTP-Headers ein, die allerdings nicht vollständig ist. In beiden Fällen ist es unmöglich, den kompletten URI zur Laufzeit zu ändern. WSDL 2.0 erfüllt zwar weitgehend die Anforderungen, wird aber nicht im BPEL unterstützt.

Das Konzept von BPEL4REST wurde dagegen mit Einbeziehung der oben genannten Anforderungen erstellt. Dementsprechend lässt diese Erweiterung sowohl die URIs als auch den HTTP-Header manipulieren und unterstützt Content-Negotiation. BPEL4REST ermöglicht es die fünf HTTP-Verben: POST, PUT, GET, DELETE und HEAD zu verwenden. Außerdem ist eine Reihe häufig verwendeter HTTP-Headers in der XML-Struktur schon vordefiniert. Das erleichtert auch dem Benutzer mit weniger detaillierten Kenntnissen des HTTP-Protokolls eine erfolgreiche Nutzung der Erweiterung.

Als einen weiteren Vorteil der BPEL4REST kann man eine einfache Installation auszeichnen. Ein Benutzer, der BPEL-Prozesse ohnehin in ODE ausführt, kann die Erweiterung einfach in ODE integrieren, ohne die bestehende Implementierung zu verändern.

3.2 Aufbau der BPEL4REST Extension Activities

BPEL ist eine erweiterbare Sprache, die das Hinzufügen neuer Aktivitäten, die nicht in der BPEL-Spezifikation definiert sind, in einen BPEL-Prozess ermöglicht. Damit neue Aktivitäten erkannt und ausgeführt werden, müssen sie sich innerhalb des XML-Elements `<extensionActivity>` befinden. Jedes `<extensionActivity>`-Element muss genau ein Kind-Element

enthalten [9]. Wenn ODE während der Ausführung eines BPEL-Prozesses ein `<extensionActivity>`-Element trifft, wird die Bearbeitung dessen Inhalts an die entsprechende Erweiterung weitergegeben.

Dieses Konzept nutzt BPEL4REST, um Aufrufe von REST Services in BPEL-Prozesse hinzuzufügen. Für jedes der unterstützten HTTP-Verben existiert ein gleichnamiges XML-Element, das ein Kind-Element des `<extensionActivity>` ist. Das Element beinhaltet sämtliche Informationen, die für die Ausführung der entsprechenden Interaktion notwendig sind.

Im Folgenden wird der Aufbau der neuen Aktivitäten anhand des XML-Elementes `<put>` erläutert und gleichzeitig werden Parallelen zum HTTP-Verb PUT dargestellt. Als Weiteres ist anzumerken, dass im Laufe dieser Arbeit teilweise nicht sinnvolle Strukturierungen der XML-Elemente in Aktivitäten der BPEL4REST festgestellt wurden. Im Folgenden wird die neue Version der BPEL4REST betrachtet und der Unterschied an geänderten Stellen erläutert.

3.2.1 Die Struktur des `<put>`-Elements

Listing 3.1: Struktur des XML-Elementes `<put>`

```
<put host="..." path="..." >
  <context ref="..." >
    ...
  </context>
  <requestParameters >
    ...
  </requestParameters>
  <responseParameters >
    ...
  </responseParameters>
</put>
```

Die gesamte Struktur des `<put>`-Elements ist im Listing 3.1 zu sehen. Die Bedeutung der Attribute und Kind-Elemente wird im Folgenden kurz erklärt.

Um die benötigte Ressource eines REST Services zu adressieren, werden die „*host*“- und „*path*“-Attribute des `<put>`-Elements verwendet. Die beiden Attribute können sowohl einen String-Wert als auch einen Verweis auf eine vordefinierte BPEL-Variable enthalten. Um die Request URI zu bilden wird der Inhalt der Attribute miteinander konkateniert.

Wie oben erwähnt wurde, beinhaltet ein `<put>`-Element nicht nur Parameter einer HTTP-Anfrage, sondern auch platzhaltende BPEL-Variablen, wo Bestandteile einer nach der Ausführung der HTTP-Anfrage ankommenden Response-Message gespeichert werden. Dementsprechend ist das `<put>`-Element folgendermaßen strukturiert; alle seine Kind-Elemente sind verpflichtend:

- `<context>` beinhaltet Elemente, die einigen general- und request-Headern entsprechen. Das optionale Attribut „*ref*“ verweist auf eine Variable, die eine Instanz von

`<context>` enthält. Es erlaubt dem Benutzer die HTTP-Headers, die in mehreren Anfragen die gleichen Werte annehmen, einmal auszufüllen. Das `<context>`-Element wird im Unterkapitel 3.2.2 genauer erklärt.

- `<requestParameters>` beinhaltet Elemente für weitere request-Headers und referenziert eine Variable, die Message-Entity enthält. Der Aufbau des Elements wird in 3.2.3 genauer erläutert.
- `<responseParameters>`: hier werden die Verweise auf Variablen, wo die Response-Message gespeichert wird, eingegeben. Das `<responseParameters>`-Element wird im Unterkapitel 3.2.4 erklärt.

3.2.2 Die Struktur des `<context>`-Elements

Listing 3.2: Struktur des `<context>`-Elements

```
<context>
  <date value="..." />
  <closeConnection value="..." />
  <authentication>
    <user>...</user>
    <pass>...</pass>
  </authentication>
  <cachecontrol>...</cachecontrol>
  <additionalHeaders>
    <header name="..." value="...">
  </additionalHeaders>
</context>
```

Das im Listing 3.2 gezeigte `<context>`-Element dient folgenden zwei Zwecken: Einerseits beinhaltet dieses Element Meta-Daten, die in mehreren Interaktionen verwendet werden. Manche dieser Daten können benutzerfreundlich und unabhängig von HTTP angegeben werden. Andererseits kann man auch noch konkrete HTTP-Header setzen.

Im Gegensatz zur ersten Version BPEL4REST wurden die XML-Elemente für request-Headers der Bedingungen (Conditionals) aus dem `<context>` ausgeschlossen, da sie in unterschiedlichen Aufrufen eines REST Services meist verschiedene Werte haben.

Alle Kind-Elemente des `<context>` sind optional und ihre Werte stehen mit folgenden HTTP-Headern der Request-Message in Zusammenhang:

- Das `<date>`-Element kann einen booleschen Wert annehmen, wobei Default auf `true` gesetzt ist. Das Element beeinflusst die Übergabe des general-Headers „Date“ einer Request-Message. Falls `<date>` mit `true` belegt ist, wird in der Request-Message der Header „Date“ mit dem Datum und der Uhrzeit ihrer Entstehung hinzugefügt.
- Das `<closeConnection>`-Element nimmt ähnlich wie das `<date>`-Element boolesche Werte an. Ist das Attribut „value“ des `<closeConnection>` auf `true` gesetzt, wird der

general-Header „Connection“ mit dem Wert „close“ in der request-Message übergeben.

- `<authentication>`: falls dieses Element vorhanden ist, müssen auch die beiden verpflichtenden Kind-Elemente angegeben werden. Die angegebenen Werte werden verwendet, um „credentials“ des request-Headers „Authorization“ zu bilden.
- Das `<cachecontrol>`-Element nimmt einen String-Wert an, der vollständig übernommen wird, um den general-Header „Cache-Control“ der Request-Message zu setzen. Demzufolge muss der hier angegebene Wert der HTTP-Spezifikation entsprechen.
- Das `<additionalHeaders>`-Element beinhaltet eine beliebige Anzahl der Kind-Elemente. Die hier angegebenen HTTP-Headers und ihre Werte werden in der Request-Message verwendet.

3.2.3 Die Struktur des `<requestParameters>`-Elements

Listing 3.3: Struktur des XML-Elementes `<requestParameters>`

```
<requestParameters>
  <contentNegotiation>
    <acceptEntity type="..." priority="..." />
    <acceptLanguage priority="..." />...</acceptLanguage>
  </contentNegotiation>
  <conditionals>
    <ifMatch>...</ifMatch>
    <ifModifiedSince>...</ifModifiedSince>
    <ifNoneMatch>...</ifNoneMatch>
    <ifRange>...</ifRange>
    <ifUnmodifiedSince>...</ifUnmodifiedSince>
  </conditionals>
  <requestEntity type="..." entity="...">
    <language>...</language>
  </requestEntity>
</requestParameters >
```

Das `<requestParameters>`-Element, das im Listing 3.3 zu sehen ist, enthält optionale Kind-Elemente, die für den Aufbau einer Request-Message zu verwendende Daten beinhalten. Im Unterschied zur ersten Version BPEL4REST findet hier das XML-Element `<conditionals>` seinen Platz. Der Zusammenhang mit der HTTP-Spezifikation wird im Folgenden erläutert.

- Das `<contentNegotiation>`-Element enthält eine beliebige Anzahl der `<acceptEntity>`- und `<acceptLanguage>`-Elemente. Die beiden Kind-Elemente werden verwendet, um Prioritäten gewünschter Media Types bzw. Languages der Response Entity anzugeben, und korrelieren mit den request-Headern „Accept“ bzw. „Accept-Language“.

- Das `<conditionals>`-Element enthält fünf optionale Kind-Elemente, die beim Aufbau „conditional GET“ eingesetzt werden und deren Inhalte in den entsprechenden request-Headern vollständig übernommen werden.
- Das `<requestEntity>`-Element wird nur bei PUT und POST verwendet, da nur in diesen zwei HTTP-Verben ein Message Body in der Request-Message sinnvoll ist. Der Inhalt der Variable, die im verpflichtenden Attribute „entity“ referenziert ist, wird in den Message Body geschrieben.

3.2.4 Die Struktur des `<responseParameters>`-Elements

Listing 3.4: Struktur des `<responseParameters>`-Elementes

```
<responseParameters>
  <responseHeader variable="..."/>
  <acceptEntityMapping type="..." variable="..."/>
  <catch status="..." faultName="..."/>
</responseParameters>
```

Das `<responseParameters>`-Element beinhaltet sowohl platzhaltende Variablen für die Informationen, die die HTTP-Antwort-Nachricht enthält, als auch ein Mapping der eventuell zurückgegebenen Status Codes auf BPEL-Fehler (siehe Listing 3.4).

- Das `<responseHeader>`-Element ist optional. In die in seinem Attribut referenzierte Variable werden Meta-Daten einer vorhandenen HTTP-Antwort gespeichert.
- Das `<acceptEntityMapping>`-Element kann beliebig oft vorkommen und wird benutzt, um in Abhängigkeit vom Media Type der im Message Body enthaltenen Entity eine bestimmte Variable zu verwenden. Das HTTP-Verb HEAD enthält dieses Element nicht, da eine HEAD-Response-Message keinen Message Body sondern nur Headers beinhaltet.
- Das `<catch>`-Element kann beliebig oft vorkommen. Dieses Element sorgt dafür, dass beim Eintreten des im Attribut „status“ eingegebenen Status-Codes ein BPEL-Fehler mit dem Namen „faultName“ ausgelöst wird.

4 Entwurf

In Kapitel 3 wurde eine bestehende Erweiterung für BPEL vorgestellt, die die Orchestrierung von REST Services ermöglicht. Um einen Aufruf eines REST Services in einen BPEL-Prozess einzubauen, müssen die Benutzer die BPEL4REST-Spezifikation kennen und XML-Abschnitte in die BPEL-Datei, die diesen BPEL-Prozess enthält, manuell hinzufügen. Der BPEL Designer, mit welchem der ganze Prozess modelliert wird, bietet keinerlei grafische Unterstützung für die `<extensionActivity>`-Elemente. Das macht die Erstellung des BPEL-Prozesses fehleranfällig und verschiebt die Entdeckung möglicher syntaktischer Fehler auf den Zeitpunkt der Ausführung des Prozesses.

Um den Benutzer beim Modellieren von REST-Aktivitäten in BPEL-Prozess grafisch zu unterstützen, wurde in dieser Arbeit eine Erweiterung der BPEL Designer konzipiert. Diese Erweiterung wird nun in Kapitel 4 beschrieben.

4.1 Allgemeine Konzepte

Beim Erstellen des Konzepts für eine Erweiterung der BPEL Designer wurden als erstes die Hauptprinzipien, die während der Entwicklung des Prototyps befolgt werden sollen, festgelegt.

- Die Erweiterung soll so weit wie möglich dem BPEL Designer ähneln, um dem Benutzer die Einarbeitungsphase zu verkürzen.
- Die Semantik der grafischen Elemente soll nicht geändert werden, um den Benutzer nicht zu verwirren.

Die BPEL4REST-Erweiterung führt die fünf neuen XML-Elemente in die BPEL-Spezifikation ein. Jedes der Elemente ist ein Kind-Element des `<extensionActivity>` und stellt damit eine Aktivität des BPEL-Prozesses dar. Im BPEL Designer wird für jede Aktivität ein grafisches Element vorbestimmt, folglich soll auch jeder der REST-Aktivitäten ein neues grafisches Element zugeteilt werden.

Wie es im Unterkapitel 2.2 erwähnt wurde, lässt der BPEL Designer mit der Drag-and-drop Technik Bestandteile eines Prozesses aus der Palette in das Prozessmodell einfügen. Dem analog sollen in die Palette des BPEL Designers Einträge für die fünf REST-Aktivitäten hinzugefügt werden. Um dem Benutzer die Suche nach Aktivitäten in der Palette zu erleichtern, sollen die neuen Paletteneinträge innerhalb der Gruppe „RESTful Activity“ platziert werden.

Beim Auswählen des grafischen Elements, das eine REST-Aktivität präsentiert, sollen Steuerelemente für die dazugehörigen Attribute und Kind-Elemente in der Ansicht „Eigenschaften“ angezeigt werden. Im Unterschied zu den meisten nativen Elementen des BPEL-Prozesses besitzt ein XML-Element der REST Activity laut seiner Spezifikation, die in 3.2 beschrieben wurde, eine umfassende Struktur mit einer großen Anzahl verschiedener Elemente,

die nicht alle gleichzeitig auf einem Reiter platziert werden können, ohne die Benutzerbarkeit zu beeinträchtigen. Um dies zu vermeiden, sind zwei verschiedene Ansätze möglich.

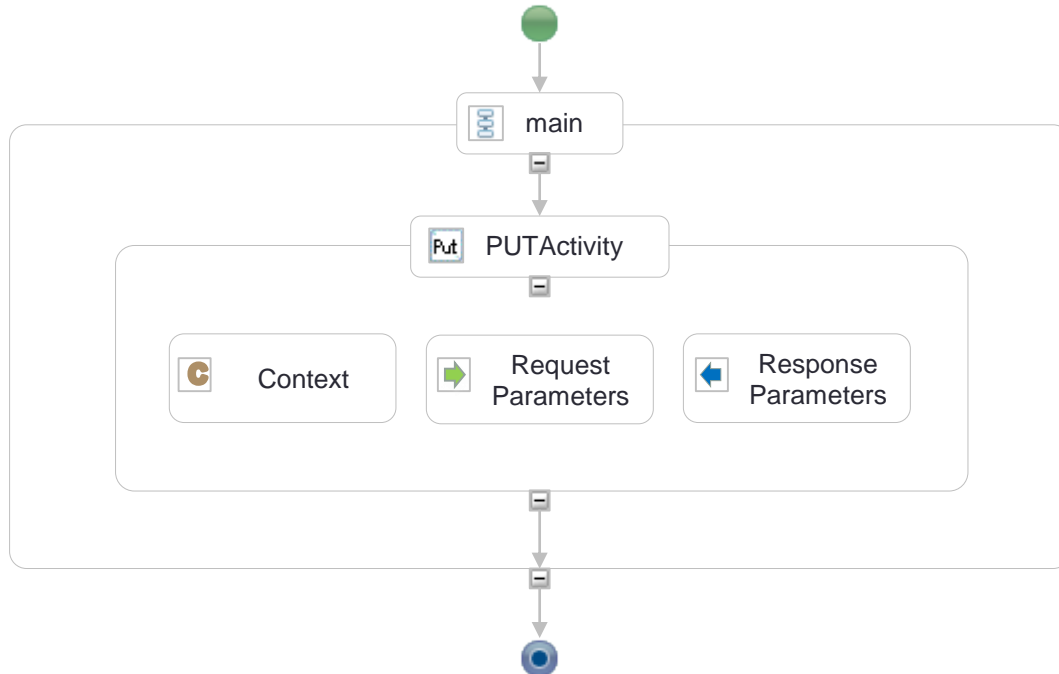


Abbildung 4.1: Eine mögliche Darstellung einer REST-Aktivität im BPEL-Prozess

Eine Möglichkeit wäre es, ein komplexeres grafisches Element, das in der Abbildung 4.1 zu sehen ist, für die REST Activity zu verwenden. Die drei untergeordneten Elemente werden für die Darstellung der `<context>`-, `<requestParameters>`- und `<responseParameters>`-XML-Elemente verwendet. Die untergeordneten Elemente sollen automatisch erstellt werden und es soll keine Möglichkeit geben, diese Elemente zu einem späteren Zeitpunkt zu entfernen.

Wird das äußere Element ausgewählt, sollen die Attribute der Aktivität am Reiter „Details“ angezeigt werden. Beim Auswählen einer der untergeordneten Elemente sollen dann auf dem Reiter „Details“ die entsprechenden Eigenschaften angezeigt werden.

Dieser Ansatz bringt Vorteile bei der Fehlerlokalisierung, da die Fehlermarker direkt an dem untergeordneten Element angezeigt werden können, welches den Fehler enthält. Besonders vorteilhaft wäre es in einer Situation, wenn eine Aktivität, die in das Prozessmodell neu eingefügt wurde, verpflichtende Kind-Elemente enthält, die unbedingt ausgefüllt werden müssen. Mit Hilfe des Markers wäre in diesem Fall ein Hinweis gegeben, wo sich die verpflichtenden und noch nicht ausgefüllten Kind-Elemente befinden.

Als Nachteil dieses Ansatzes kann man anmerken, dass aus der semantischen Sicht ein Widerspruch zu anderen Elementen des BPEL-Prozesses entsteht. Im BPEL Designer sind alle untergeordneten Elemente einer komplexen Aktivität jeweils auch Aktivitäten, die in einer bestimmten Reihenfolge durchgeführt werden. In unserem Fall aber stellen die untergeordneten Elemente keine Aktivitäten dar, sondern werden nur für eine Gruppierung der Eigenschaften verwendet.

Eine weitere Möglichkeit wäre, für die Darstellung einer REST-Aktivität ein gewöhnliches grafisches BPEL-Activity-Element zu verwenden und die Steuerelemente für die zahlreichen Eigenschaften einer REST-Aktivität zwischen verschiedenen Reitern zu verteilen. Dabei sollen die drei zusätzlichen Reiter in die Ansicht „Eigenschaften“ hinzugefügt werden.

Dieser Ansatz bietet eine platzsparende Darstellung der REST-Aktivität und kurze Wege beim Umschalten zwischen den Reitern. Außerdem würde bei einer eventuellen Weiterentwicklung und Änderung des Designs des BPEL Designers auch die Darstellung der REST-Aktivitäten automatisch angepasst werden, da hier ein natives BPEL-Activity-Element verwendet wird.

Als ein Nachteil des zweiten Ansatzes lässt sich feststellen, dass die Fehler nicht so gut lokalisiert werden, wie beim ersten Ansatz. Allerdings ist dies kein kritischer Entscheidungspunkt, da in der aktuellen Spezifikation der BPEL4REST keine verpflichtenden Kind-Elemente der <context>, <requestParameters> und <responseParameters> vorhanden sind. Die zusätzlichen Datenvalidierungsmechanismen, die eingebaut werden sollen, werden diesen Nachteil ausgleichen und die gesamte Benutzerbarkeit erhöhen.

Auf Grund der obengenannten Überlegungen wurde die zweite Möglichkeit gewählt. Die Belegung der Reiter mit Steuerelementen und deren Zusammenhang mit den XML-Elementen der BPEL4REST wird im Unterkapitel 4.2 erklärt.

4.2 UI Prototyp

In diesem Abschnitt wird die Platzierung der Steuerelemente auf die Reiter genauer beschrieben. Die Erklärung erfolgt am Beispiel der REST-Aktivität „Put“. Außerdem wird auf Unterschiede mit weiteren REST-Aktivitäten hingewiesen.

The image shows a screenshot of the BPEL Designer's user interface. At the top, there are three tabs: 'Properties', 'Problems', and '...'. The main area is titled 'ExtensionActivity Put'. On the left side, there is a vertical list of tabs: 'Description', 'Details' (which is selected and highlighted), 'Join Behaviour', 'Namespaces', 'Context', 'Request Parameters', 'Response Parameters', and 'Documentation'. The 'Details' tab is active, showing two input fields. The first field is labeled 'Host:' and contains the text 'text'. The second field is labeled 'Path:' and contains the text '\$variableRef\$'. Both input fields have a small '...' button to their right, indicating that the content can be expanded or edited.

Abbildung 4.2: Belegung des Reiters "Details"

4.2.1 Reiter „Details“

Auf dem Reiter „Details“ finden die Steuerelemente für die Attribute der Put-Aktivität, die im Listing 3.1 zu sehen sind, ihren Platz. Die prototypische Darstellung des Reiters kann der Abbildung 4.2 entnommen werden.

Da die beiden Attribute sowohl einen String-Wert annehmen können, als auch auf eine BPEL-Variable verweisen, werden den Benutzern zwei Möglichkeiten, die Felder zu editieren, angeboten. Einerseits ist es möglich, direkt im Textfeld einen String-Wert einzugeben. Andererseits wird beim Drücken auf die Schaltfläche ein Auswahldialog geöffnet, der dem Benutzer das Referenzieren einer BPEL-Variablen ermöglicht.

4.2.2 Reiter „Context“

The screenshot shows the configuration interface for an 'ExtensionActivity Put'. The 'Context' tab is active, displaying various settings. The 'Context ref' field is set to '\$variableRef\$'. The 'Authentication' section is checked and includes 'User' and 'Password' text boxes. The 'Cache Control' section has a text box with 'text'. The 'Additional Headers' section is also checked and contains a table with three headers and their values. The table is as follows:

Header Name	Value
Header 1	Value 1
Header 2	Value 2
Header 3	Value 3

Abbildung 4.3: Belegung des Reiters "Context"

Die Steuerelemente für Kind-Elemente des <context>-Elements (Listing 3.2) werden auf dem gleichnamigen Reiter „Context“ platziert, dessen prototypische Darstellung in der Abbildung 4.3 angezeigt wird. Der „Context“-Reiter wird für alle Typen der REST-Aktivitäten gleich aufgebaut.

Für das Einfügen einer Variablen in das Feld „Context ref“ muss die Schaltfläche betätigt werden, dabei wird ein Auswahldialog geöffnet. Das direkte Editieren des Feldes ist untersagt, weil die Textwerte, die keinen Bezug auf eine deklarierte Variable haben, zu Fehlern während der Ausführung führen werden. Der Name einer in diesem Feld ausgewählten Variablen wird in das Attribut „ref“ geschrieben.

Beim Einschalten eines der Auswahlkästchen wird im <context>-Element ein entsprechendes Kind-Element hinzugefügt. Im Falle der „Authentication“ und „Additional Headers“ werden die weiteren zur jeweiligen Gruppe gehörenden Steuerelemente für eine Editierung aktiviert.

Die „Additional Headers“-Tabelle ist für das Hinzufügen einer beliebigen Anzahl von Headers vorbestimmt. Beim Betätigen der Schaltfläche „New Header...“ wird ein Dialog geöffnet, der zur Erstellung eines neuen Headers dient. Das Hinzufügen mehrerer Headers mit demselben Namen ist nicht erlaubt.

Der Inhalt des „Cache Control“-Feldes ist unmittelbar mit dem <cachecontrol> verbunden. Beim Hinzufügen eines Textwertes in dieses Feld wird das XML-Element erstellt, falls es noch nicht existiert, oder dessen Inhalt geändert. Das komplette Löschen des Inhaltes des „Cache Control“-Feldes hat das Entfernen des <cachecontrol>-Elements aus dem <context> zur Folge.

4.2.3 Reiter „Request Parameters“

The screenshot shows the 'Request Parameters' tab in an IDE. The main area is titled 'ExtensionActivity Put'. It contains several sections for configuring request parameters:

- Content Negotiation:**
 - Content Negotiation
 - Accept Entities:**

Media Type	Priority	
Media Type 1	Priority 1	
Media Type 2	Priority 2	
 - Accept Languages:**

Language	Priority	
Language 1	Priority 1	
Language 2	Priority 2	
- Conditionals:**
 - Conditionals
 - Match: text
 - Modified Since: date
 - None Match: text
 - Range: text/date
 - Unmodified Since: date
- Request Entity:**
 - Request Entity
 - Type: text
 - Entity: \$variableRef\$
 - Language: text

Abbildung 4.4: Belegung des Reiters "Request Parameters"

Der in der Abbildung 4.4 gezeigte Reiter „Request Parameters“ wird mit Steuerelementen, die für Kind-Elemente des <requestParameters>-Elements (Listing 3.3) vorgesehen sind, belegt. Dieser Reiter unterscheidet sich bei der Anzeige verschiedener REST-Aktivitäten. Nur

die „Request Parameters“-Reiter für die Put- und Post-Aktivitäten enthalten die Gruppe „Request Entity“, was der Spezifikation von BPEL4REST entspricht.

Analog dem Reiter „Context“ dienen die Auswahlkästchen der Aktivierung bzw. Deaktivierung der zugehörigen Steuerelemente. Außerdem sind die Auswahlkästchen für das Hinzufügen und das Löschen der entsprechenden Kind-Elemente im XML-Element `<requestParameters>` zuständig.

Die beiden Tabellen „Accept Entities“ und „Accept Languages“ erlauben, eine beliebige Anzahl der Einträge hinzuzufügen. Um dem Benutzer eine bessere Übersicht zu gewährleisten, werden die Zeilen der Tabellen nach der Spalte „Priority“ geordnet, dabei werden die am meisten bevorzugten Media Types bzw. Languages oben in der Tabelle platziert.

Beim Betätigen der Schaltfläche „New Entity...“ bzw. „New Language...“ wird ein Dialogfenster geöffnet, das jeweils zwei Auswahllisten enthält. Eine prototypische Darstellung ist in der Abbildung 4.5 zu sehen. Die Erste der Listen enthält einige Werte der Media Types bzw. Languages und ist editierbar. Wegen einer großen Anzahl der möglichen Werte ist es nicht sinnvoll alle aufzulisten. Der Benutzer hat die Möglichkeit einen Wert sowohl aus der Liste zu wählen, als auch manuell in das Feld zu schreiben. Im Gegenteil dazu bietet die Priority-Liste nur Auswahlmöglichkeiten. Dies schützt den Benutzer vor inkorrekten Angaben.

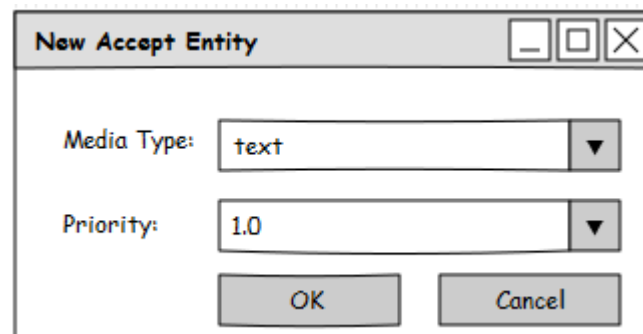


Abbildung 4.5: Dialogfenster für das Hinzufügen einer neuen Accept Entity

Der Inhalt der Steuerelemente der Gruppe „Conditionals“ ist unmittelbar mit den gleichnamigen XML-Elementen der BPEL-Datei verbunden. BPEL4REST stellt keine Anforderungen an die Kind-Elemente des `<conditionals>`-Elements, abgesehen davon, dass die Kind-Elemente einen String-Wert beinhalten sollen. In diesem Fall beeinflusst die HTTP-Spezifikation die Weise, auf welche die Steuerelemente der Gruppe „Conditionals“ editiert werden können. „If-Match“ und „If-Non-Match“ sind zum Editieren vorbestimmt. „If-Modified-Since“ und „If-Unmodified-Since“ sind nicht editierbar, lassen jedoch mit Hilfe einer vorgesehenen Schaltfläche ein Datum aus einem Dialog auswählen, da gemäß der HTTP-Spezifikation die beiden HTTP-Headers ein Datum im Format „HTTP -date“ enthalten müssen. In das Steuerelement „Range“ kann nicht nur ein String-Wert manuell eingetragen werden, sondern auch aus einem Dialog ein Datum ausgewählt.

Schließlich sind in der Gruppe „Request Entity“ eine editierbare Auswahlliste für Media Types und ein nicht editierbares Feld mit einer Schaltfläche für Variablenauswahl enthalten.

Die hier eingegebenen Werte hängen mit den Attributen des `<requestEntity>`-Elements zusammen. Außerdem lässt die editierbare Auswahlliste „Language“ ein gleichnamiges Kind-Element des `<requestEntity>` hinzufügen, editieren und löschen.

4.2.4 Reiter „Response Parameters“

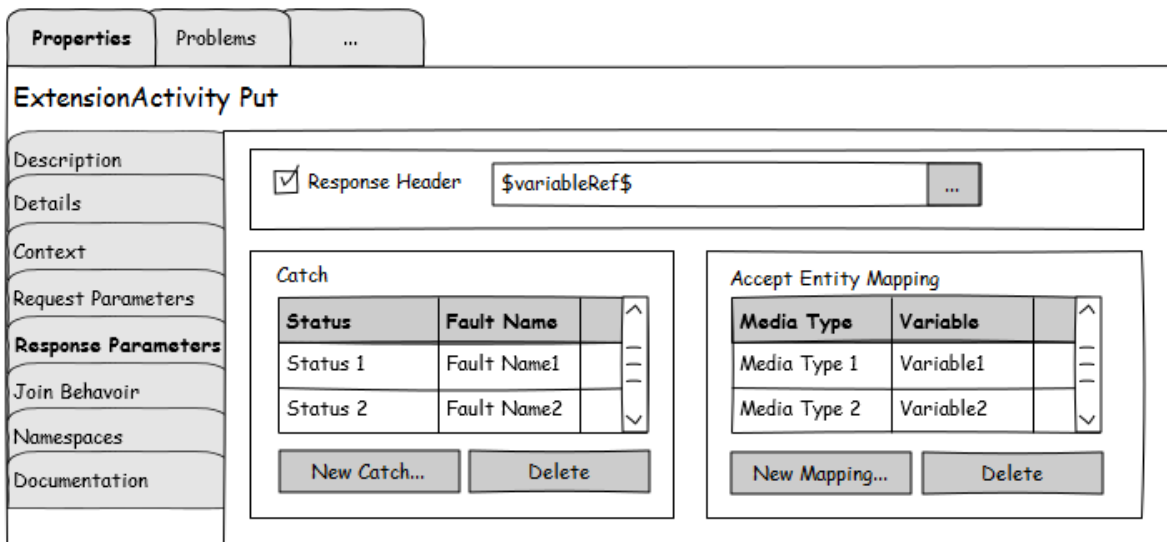


Abbildung 4.6: Belegung des Reiters "Response Parameters"

In der Abbildung 4.6 ist eine prototypische Darstellung des Reiters „Response Parameters“ zu sehen, dessen Steuerelemente sich auf die Kind-Elemente des `<responseParameters>`-Elements (Listing 3.4) beziehen. Im Falle der HEAD-Aktivität werden Steuerelemente der „Accept Entity Mapping“-Gruppe nicht angezeigt.

Das Auswahlkästchen „Response Header“ ist für das Hinzufügen und das Löschen des XML-Elements verantwortlich. Das Editieren des dazugehörigen Attributes „variable“ erfolgt im nebenstehenden Textfeld, das den Namen einer referenzierten Variablen anzeigt. Das Bearbeiten des Feldes erfolgt durch das Betätigen der vorhandenen Schaltfläche. Dabei ist das manuelle Editieren untersagt, um eventuelle Verweise auf eine nicht existierende Variable zu vermeiden.

Die beiden Tabellen sind auf dem gleichen Prinzip aufgebaut wie auch die Tabellen der anderen Reiter. Um ein `<catch>`-Element in ein BPEL-Prozess einzufügen, wird die Schaltfläche „New Catch...“ betätigt. Das Dialogfenster beinhaltet ein Textfeld zum manuellen Editieren, in welchem der Benutzer einen Status-Code eintragen kann. Außerdem wird eine Auswahlliste zur Verfügung gestellt. Diese Auswahlliste enthält Namen der in sichtbaren FaultHandlers behandelten BPEL-Fehler, kann jedoch manuell editiert werden.

Dem analog wird ein neues `<acceptEntityMapping>`-Element eingefügt. Das Dialogfenster für das Erstellen dieses XML-Elementes beinhaltet eine editierbare Auswahlliste mit Media Types und ein Textfeld für die Auswahl einer Variablen, deren Namen in das Attribut „variable“ geschrieben wird.

4.3 Validierung

Um den Benutzer bei der Erstellung eines BPEL-Prozesses zu unterstützen, wird die Datenvalidierung implementiert, die der Fehlervermeidung dient und die Fehlersuche erleichtert. Der Benutzer kann sowohl den Prozess anhand des grafischen Editors modellieren, als auch den Quelltext der BPEL-Datei manuell editieren. Aus diesem Grund werden verschiedene Arten der Datenvalidierung eingesetzt.

Einerseits sollte die Kontrolle beim Betätigen der verschiedenen Steuerelemente in der grafischen Ansicht durchgeführt werden. Es wird bei den „OK“-Bestätigungen in den Dialogfenstern für die Erstellung neuer Tabelleneinträge ein Hinweis gegeben, wenn in Form vorgegebene Eingabefelder nicht vollständig ausgefüllt sind oder in Tabellen doppelte Einträge entstehen. Außerdem werden verpflichtende Textfelder, die keine Eingabe enthalten, markiert, um den Benutzer darauf aufmerksam zu machen.

Andererseits sollte eine Validierung des Quelltextes der BPEL-Datei stattfinden, die beim Betätigen des Eintrages des Kontextmenüs „Validate“ aufgerufen werden kann. Die XML-Elemente der BPEL-Datei, die der BPEL4REST-Spezifikation nicht entsprechen, werden markiert. Dabei wird überprüft, ob XML-Elemente alle verpflichtenden Attribute und Kind-Elemente haben, referenzierte Variablen im BPEL-Prozess deklariert sind und der Typ der Attribute mit der BPEL4REST-Spezifikation übereinstimmt. Die bei der Validierung generierten Marker werden zusätzlich im grafischen Editor angezeigt.

5 Implementierung

In Kapitel 5 wird die Implementierung der Erweiterung für BPEL Designer, im Folgenden *RESTModelingExtension* genannt, beschrieben. Zuerst werden die bei der Entwicklung und dem Testen verwendeten Technologien vorgestellt, danach auf die Besonderheiten der Plugins-Entwicklung eingegangen und zum Schluss die Architektur der *RESTModelingExtension* erläutert.

5.1 Entwicklungs- und Testumgebung

Im Unterkapitel 5.1 werden Werkzeuge und Technologien kurz vorgestellt, die für die Entwicklung und das Testen eingesetzt worden sind.

5.1.1 Eclipse

Eclipse ist eine Open-Source-Software, für deren Entwicklung Eclipse Foundation verantwortlich ist [15]. Für die Entwicklung wurde die *Version Eclipse IDE for Java EE Developers Helios Service Release 2* verwendet.

Die Installationspakete beinhalten folgende Projekte, die für die Entwicklung der *RESTModelingExtension* eingesetzt wurden:

- Eclipse Modeling Framework (EMF) ist ein Open-Source Framework für die Modellierung und automatisierte Generierung des Codes aus den entworfenen Modellen [16].
- Graphical Editing Framework (GEF) bietet Technologien für die Erstellung grafischer Editoren an [17].

5.1.2 SimTech BPEL Designer

Die Entwicklung und das Testen der *RESTModelingExtension* wurde unter Verwendung des SimTech BPEL Designers durchgeführt. Dieser Designer erweitert die Möglichkeiten der Eclipse BPEL Designer. Die Besonderheiten von SimTech BPEL sind in Kapitel 2.2.2 erläutert.

Für Installation von SimTech BPEL Designer benötigt man zusätzliche Komponenten [18]:

- Eclipse Modeling Framework (EMF) Compare SDK erweitert EMF um einen Modellvergleich und bietet generische Unterstützung für alle Arten der Metamodelle [19].
- Graphical Modeling Framework (GMF) Runtime ist ein Anwendungs-Framework für die Erstellung grafischer Editoren unter der Nutzung von EMF und GEF [20].

- Apache Tomcat ist ein Open-Source Servlet/JSP Container, der die Java Servlet und JavaServer Pages implementiert und weitere Features anbietet, die ein Deployment von Web-Anwendungen und Web Services ermöglichen [21]. SimTech BPEL Designer verwendet Apache Tomcat als einen Container für ODE, wo die BPEL-Prozesse ausgeführt werden. Die verwendete Version ist 7.0.34
- Apache ActiveMQ ist eine Open-Source Implementierung des Java Message Services 1.1 (JMS) und wird als Message Broker bezeichnet [22]. SimTech BPEL Designer verwendet für Monitoring laufender BPEL-Instanzen die Version 5.7.0

5.2 Einbettung der RESTModelingExtension in den BPEL Designer

Quelle: [12]

Eclipse ist eine erweiterbare Plattform, die die Entwicklung neuer Tools mittels anschließbarer Komponenten (Plug-ins) ermöglicht. Der grundlegende Mechanismus der Erweiterbarkeit in Eclipse besteht in der Tatsache, dass ein neuer Plug-in seine Elemente zu existierenden Plug-ins hinzufügen kann, wenn die Letzten eine oder mehrere Extension-Points zur Verfügung stellen. Dabei bietet Eclipse als Ausgangspunkt eine Anzahl von Plug-ins, wie z.B. WorkbenchPlugin [23].

Die RESTModelingExtension ist keine Einzelkomponente, sondern besteht aus drei Eclipse Plug-ins, die sich bestehenden Eclipse bzw. BPEL Designer Plug-ins anschließen. Wie die Abbildung 5.1 illustriert, offerieren Komponenten von BPEL Designer verschiedene Extension-Points, an welche die RESTModelingExtension ihre Erweiterungen anbindet:

- **uiObjectFactories:** Der Extension-Point `uiObjectFactories` enthält eine Liste der Fabrik-Klassen, die grafische Objekte bestimmter Typen erstellen können. Die hier aufgelisteten Klassen müssen von der `AbstractUIObjectFactory`-Klasse erben. Die RESTModelingExtension schließt an diesen Extension-Point die Klasse `RESTActivityUIObjectFactory` an, die für die Erstellung neuer grafischer Objekte für REST-Aktivitäten benutzt wird.
- **paletteAdditions:** Der Extension-Point `paletteAdditions` ermöglicht das Hinzufügen neuer Einträge in die Palette des BPEL Designers. Dabei kann man entweder eine Liste mit Einträgen für jedes neue Element der Palette erstellen oder einen Provider definieren, der das `IPaletteProvider`-Interface implementieren muss und mehrere Einträge in die BPEL-Palette hinzufügen kann. In der RESTModelingExtension wird eine Provider-Klasse `RESTActivityPaletteProvider` definiert, die das Hinzufügen der neuen Einträge für die REST-Aktivitäten ermöglicht.
- **factories:** Der Extension-Point `factories` registriert zusätzliche Fabrik-Klassen für die Validierung im BPEL-Validator. Die hier aufgelisteten Klassen müssen das `IValidator`-Interface implementieren. Die RESTModelingExtension registriert in diesem Extension-Point eine `RESTFactory`-Klasse, die die Validatoren für alle Elemente der BPEL4REST-Erweiterung erstellen kann.

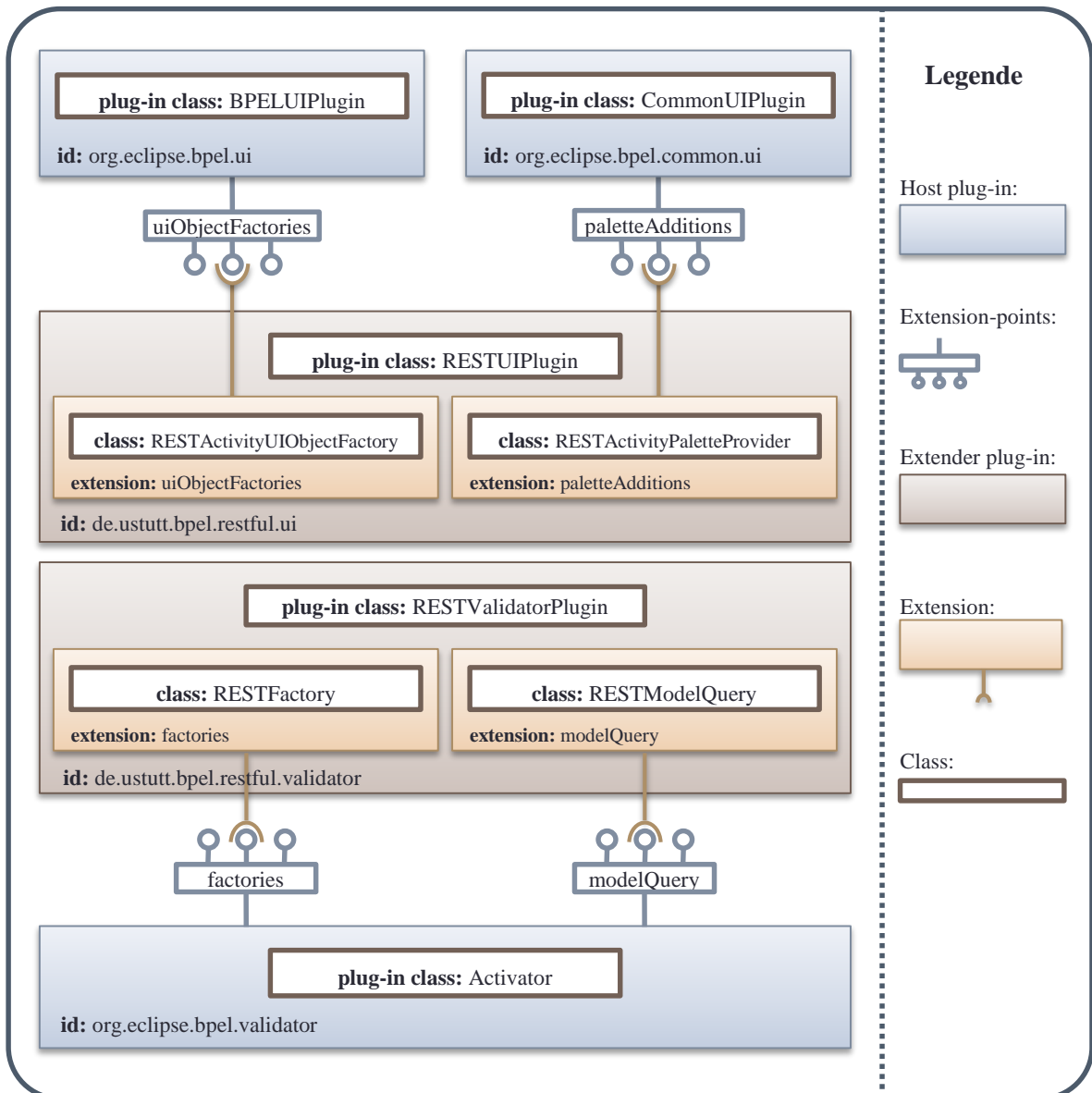


Abbildung 5.1: Beziehungen zwischen den RESTModelingExtension-Plug-ins und Plug-ins vom BPEL Designer

- modelQuery:** Der Extension-Point modelQuery enthält eine Liste mit Klassen, die der Validator benutzt, um Elemente des zu validierenden Modells abzufragen. Um dem Validator Zugriff auf Modellelemente zu gewährleisten, müssen die Query-Klassen *IModelQuery*-Interface implementieren.

Um dem Benutzer einen Zugriff auf Eigenschaften der REST-Aktivitäten zu ermöglichen, verwendet die RESTModelingExtension die Extension-Points eines von Eclipse angebotenen Plug-Ins, nämlich *TabbedPropertyViewPlugin*:

- **propertyTabs:** Der Extension-Point `propertyTabs` beschreibt eine Liste der Reiter, die die Ansicht „Eigenschaften“ unterstützen soll. In der Ansicht „Eigenschaften“ werden für REST-Aktivitäten die drei zusätzlichen Reiter angezeigt: „Context“, „Request Parameters“ und „Response Parameters“.
- **propertySections:** Der Extension-Point `propertySections` enthält eine Liste der Sektionen, die auf den Reitern der Ansicht „Eigenschaften“ angezeigt werden. Für jede Sektion muss eine Klasse, die für die Zusammensetzung der Steuerelemente auf einem Reiter verantwortlich ist, angegeben werden. Die Klasse muss von *AbstractPropertySection*-Klasse erben. Außerdem wird ein Verweis auf den Reiter angegeben, bei dem diese Steuerelemente angezeigt werden sollen, und eine Reihe von Klassen, für Eigenschaften deren Objekte die Steuerelemente vorbestimmt sind. Die Sektionen, welche durch die *RESTModelingExtension* in die Reiter der Ansicht „Eigenschaften“ hinzugefügt werden, sind im Unterkapitel 5.3.3 genauer beschrieben.

5.3 Architektur

Im Unterkapitel 5.3 werden Beziehungen zwischen den drei Plug-ins von *RESTModelingExtension* beschrieben. Zuerst wird eine Übersicht der Architektur vorgestellt und danach jede Komponente genauer erklärt.

5.3.1 Übersicht der Architektur

Die *RESTModelingExtension* teilt Zuständigkeiten zwischen den Plug-ins auf (siehe Abbildung 5.2). Die *de.ustutt.bpel.restful.ui* und *de.ustutt.bpel.restful.model* sind wesentliche Teile der Erweiterung, die den im Kapitel 4.2 vorgestellten UI Prototyp realisieren und dementsprechend dem Benutzer das Hinzufügen der REST-Aktivitäten in einen BPEL-Prozess und ebenfalls in eine zugehörige BPEL-Datei ermöglichen. Der *de.ustutt.bpel.restful.validator* Plug-in hat keine direkte Verbindung mit den anderen beiden Plug-ins und dient zur Validierung der entstandenen BPEL-Datei in Bezug auf die Konformität mit der BPEL4REST-Spezifikation. Die Beschreibung der Validierungs-Komponente ist im Unterkapitel 5.3.5 zu sehen.

Die Realisierung des UI Prototyps basiert auf dem Architekturmuster Model-View-Controller. Die Zuständigkeiten werden zwischen den drei Komponenten geteilt. Die **Model**-Komponente enthält Daten, die während einer Anwendungs-Session persistent bleiben. Die Model-Komponente benachrichtigt die Controller-Komponente, wenn die Daten geändert wurden.

Die **View**-Komponente ist für die Darstellung der sichtbaren Elemente, wie z.B. eine der REST-Aktivitäten im BPEL-Prozess sowie die Steuerelemente für ihre Attribute in der Ansicht „Eigenschaften“, und für die Benachrichtigung der Controller-Komponente, falls der Benutzer mit Elementen agiert, zuständig.

Die **Controller** Komponente verbindet die beiden Anderen, indem sie die Model-Komponente und die View-Komponente beobachtet und synchronisiert, wenn Objekte einer der Komponenten geändert wurden. Die einzelnen Komponenten sind in den Unterkapiteln 5.3.2, 5.3.3 und 5.3.4 beschrieben.

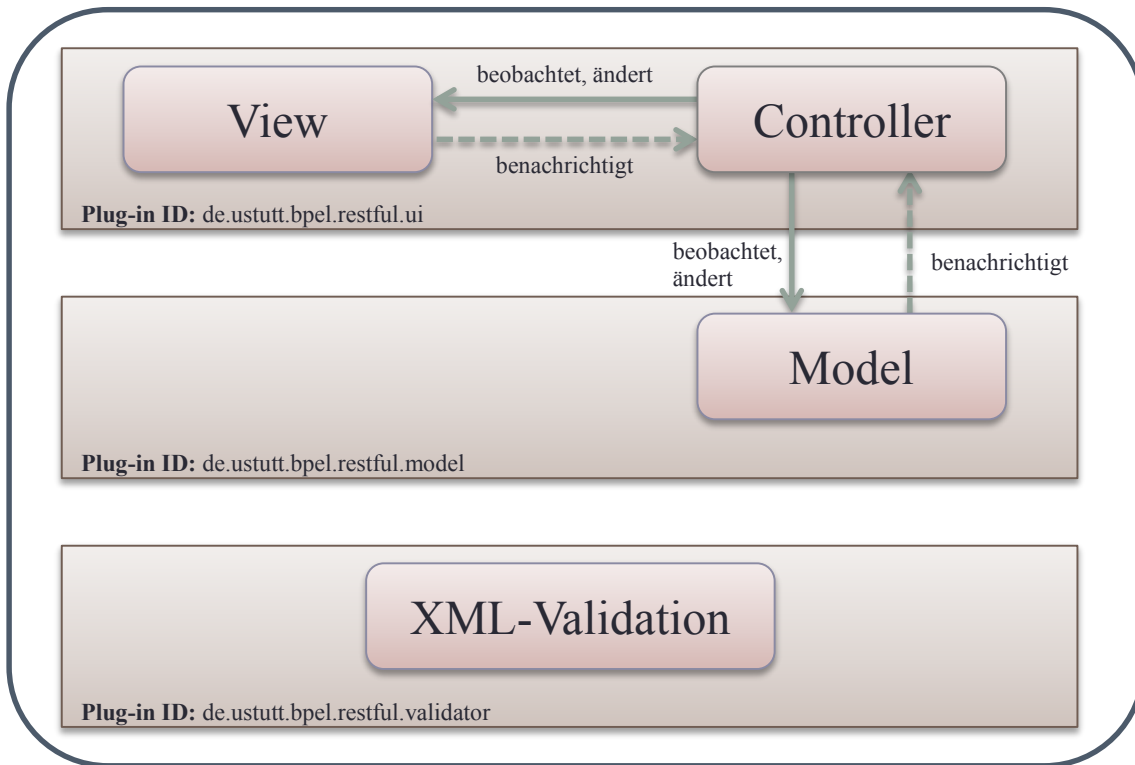


Abbildung 5.2: Übersicht der Architektur

5.3.2 Model-Komponente

Wie es im Kapitel 2.2.1 erwähnt wurde, lässt der BPEL Designer einen BPEL-Prozess sowohl in grafischer als auch in textueller Form einsehen und bearbeiten. Dies wird dadurch ermöglicht, dass der BPEL Designer immer zwei Modelle eines BPEL-Prozesses hat. Die grafische Ansicht basiert auf einem EMF-Modell, die Textansicht auf einem XML/DOM-Editor. Diese beiden Modelle müssen stets synchronisiert sein.

Ebenfalls folgt die Erweiterung demselben Prinzip. Die RESTModelingExtension verwendet nämlich ein EMF-basiertes Modell, das die BPEL4REST-Spezifikation repräsentiert und auf dem Meta-Modell für REST Extension Activity basiert [1], um Datenpersistenz zu gewährleisten und die Benachrichtigungen bei Änderungen im Modell sofort zu erhalten. Für die textuelle Repräsentation benutzt die RESTModelingExtension einen XML/DOM-Editor des BPEL Designers ohne weitere Änderungen. Die Model-Komponente sorgt außerdem für die Synchronisation der beiden Darstellungsformen.

Für jedes XML-Element aus der BPEL4REST-Erweiterung existiert in der Model-Komponente eine Klasse, die Informationen über alle Kind-Elemente und Attribute beinhaltet. Ein Ausschnitt aus dem Klassendiagramm ist in der Abbildung 5.3 dargestellt.

Alle *RestActivity*-Klassen erweitern die Klasse *ExtensionActivity* des BPEL Designers. Die Vererbungskette entwickelt sich im Zusammenhang mit Request und Response Parametern. So erben die Put- und Post- Aktivitäten, die zusätzlich eine *RequestEntity* als Request-Parameter haben, von der abstrakten Klasse *InteractionWithRequestEntity*. Im Unterschied zu

den drei anderen Aktivitäten, bei denen dem Attribut `requestParameters` ein Objekt der Klasse `RequestParametersBase` zugewiesen wird, wird in `InteractionWithRequestEntity` stets eine Instanz der Klasse `RequestParametersExtended` referenziert. Dem analog unterscheidet sich die Klasse `HeadActivity` von allen `InteractionWithEntityMapping`-Klassen, die als `responseParameter` eine Referenz auf ein Objekt der Klasse `ResponseParametersExtended` zugewiesen bekommen.

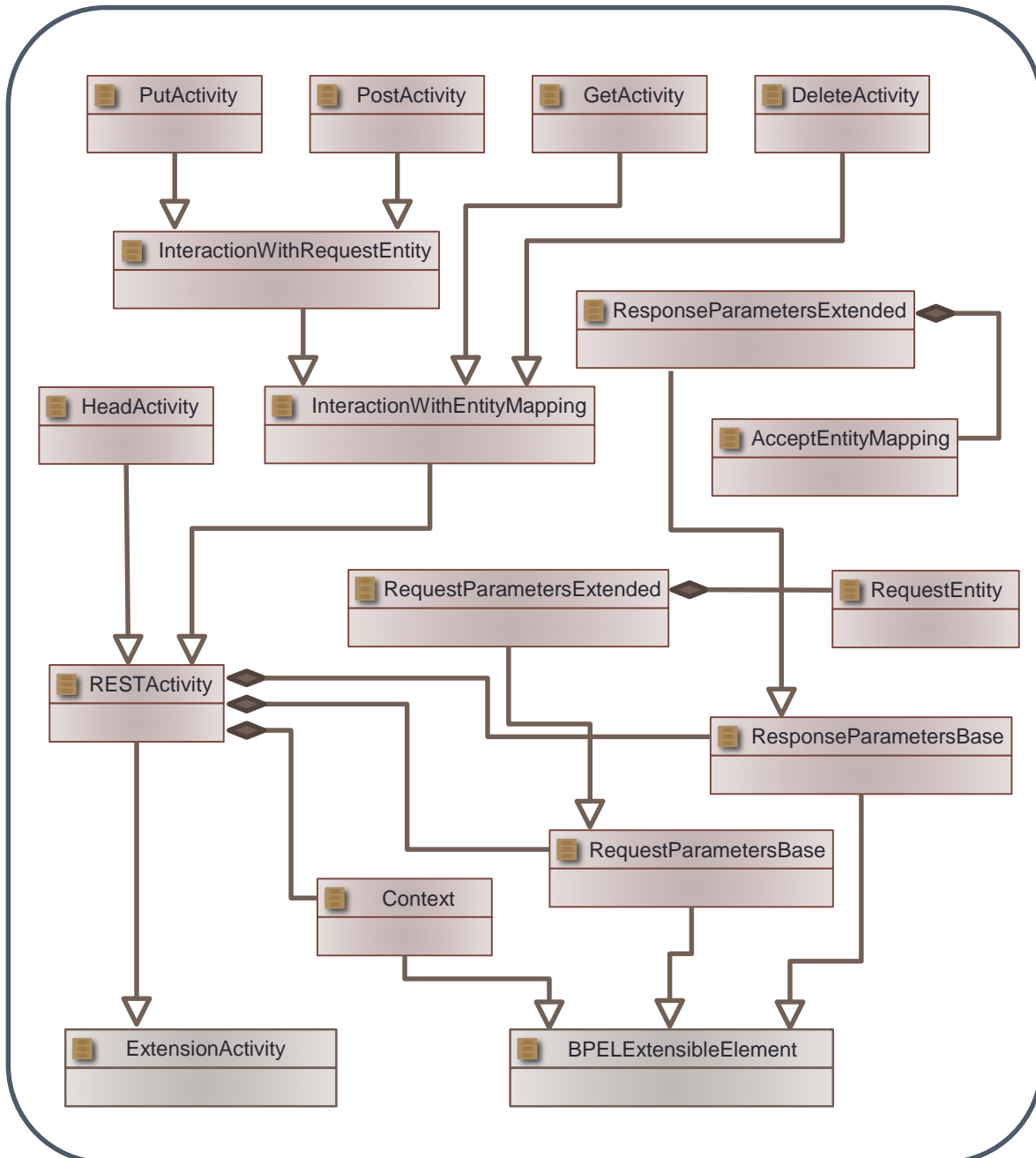


Abbildung 5.3: Ausschnitt aus dem Klassendiagramm des EMF-Modells

In der Model-Komponente spielen die zwei folgenden Klassen, die das Entwurfsmuster Singleton realisieren, eine zentrale Rolle:

- **ModelPackage** beinhaltet Informationen über alle im Modell vorhandenen Klassen mit ihren Attributen und Referenzen.
- **ModelFactory** wird für die Erzeugung neuer Instanzen der Modellklassen benutzt.

Die Controller-Komponente verwendet ModelPackage und ModelFactory, um mit Objekten des EMF-Modells zu agieren.

Eine weitere Aufgabe der Model-Komponente ist die Synchronisierung des EMF-Modells mit seiner BPEL-Darstellung. Die Synchronisierung soll erfolgen, wenn ein Benutzer eine neue REST-Aktivität aus der Palette in den BPEL-Prozess hinzufügt, mit Hilfe des Auswahlkästchens die Editierung einer Eigenschaft anschaltet oder anhand der vorhandenen Steuerelemente eine Eigenschaft editiert. Außerdem sollte bei manuellen Änderungen der BPEL-Datei die grafische Darstellungsform angepasst werden.

Die Übereinstimmung zwischen Elementen der zwei Darstellungsformen wird durch Transformationen beschrieben. Analog den in der Abbildung 5.4 dargestellten Transformationen für eine Put-Aktivität werden auch die anderen REST-Aktivitäten transformiert. Die REST-ModelingExtension verwendet die folgenden Klassen, um die beiden Darstellungsformen zu synchronisieren:

- **RestActivitySerializer** fügt in die BPEL-Datei entsprechende XML-Elemente für REST-Aktivitäten hinzu, wenn ein Benutzer eine neue REST-Aktivität im BPEL-Prozess erstellt.
- **ChildElementSerializer** fügt in ein REST-Aktivitätselement der BPEL-Datei entsprechende Kind-Elemente ein, wenn ein Benutzer eine neue Eigenschaft von REST-Aktivität mit Hilfe eines Auswahlkästchens einschaltet.
- **RestActivityDeserializer** ist zuständig für die Synchronisierung der Modelldaten mit den Inhalten der BPEL-Datei. Die Klasse wird bei jeder Änderung in der BPEL-Datei aufgerufen.
- **RESTRconciliationHelper** ist eine Klasse die von *ReconciliationHelper* des BPEL Designers erbt. Wenn ein Benutzer eine Eigenschaft einer der REST-Aktivitäten im BPEL-Prozess ändert, sollte auch der Inhalt der BPEL-Datei überschrieben werden. Dazu wird bei jeder Änderung der Modellattribute die statische Methoden **replaceAttribute()**, **replaceChild()**, **addTextChild()**, **removeTextChild()** der Klasse *RESTRconciliationHelper* aufgerufen.

Bei der Initialisierung des ModelPackages werden die beiden Serializer und der Deserializer im *BPELExtensionRegistry* registriert, damit der BPEL Designer Zugriff auf diese Klassen hat. *BPELExtensionRegistry* erbt von *javax.wsdl.extensions.ExtensionRegistry* und wird benutzt, um Serializer und Deserializer mit erweiterbaren Elementen zu assoziieren.

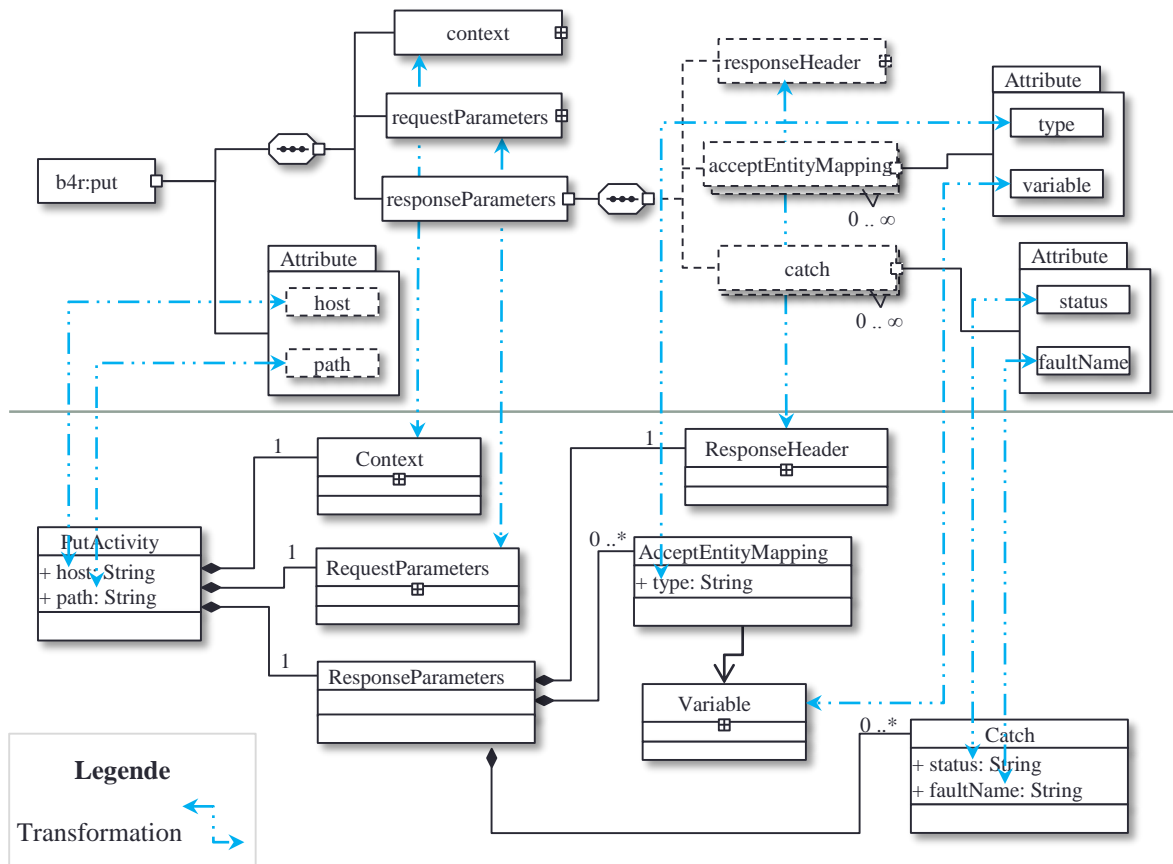


Abbildung 5.4: Transformationen zwischen den XML-Elementen und Klassen des EMF-Modells

5.3.3 View-Komponente

Die RESTModelingExtension wie auch der BPEL Designer selbst, benutzt sowohl GEF als auch SWT Technologien, um grafische Objekte darzustellen. Dementsprechend kann man alle Objekte der View-Komponente, die für das Darstellen der sichtbaren Elemente verantwortlich sind, in zwei Gruppen unterteilen. Einerseits sind das die Draw2D Figuren für die Darstellung der REST-Aktivitätselemente im BPEL-Prozess, die auf einem SWT Canvas angezeigt werden. Andererseits sind das SWT Controls, die für die Darstellung der Elemente in Reitern der Ansicht „Eigenschaften“ verwendet werden.

RESTModelingExtension setzt für die REST-Aktivitätselemente die Figuren des BPEL Designers ein, die in den entsprechenden EditParts instanziiert werden. Alle Draw2D Figuren haben keine Kenntnisse über Model- oder Controller-Objekte. SWT Canvas benachrichtigt die registrierten Beobachter bei Aktivitäten des Benutzers.

SWT Controls werden in PropertySection-Klassen erstellt. Sie kennen zwar die zugehörigen Model-Objekte nicht, senden aber selbst die Benachrichtigungen zu den registrierten Beobachtern, wenn ein Ereignis von der Benutzerseite kommt.

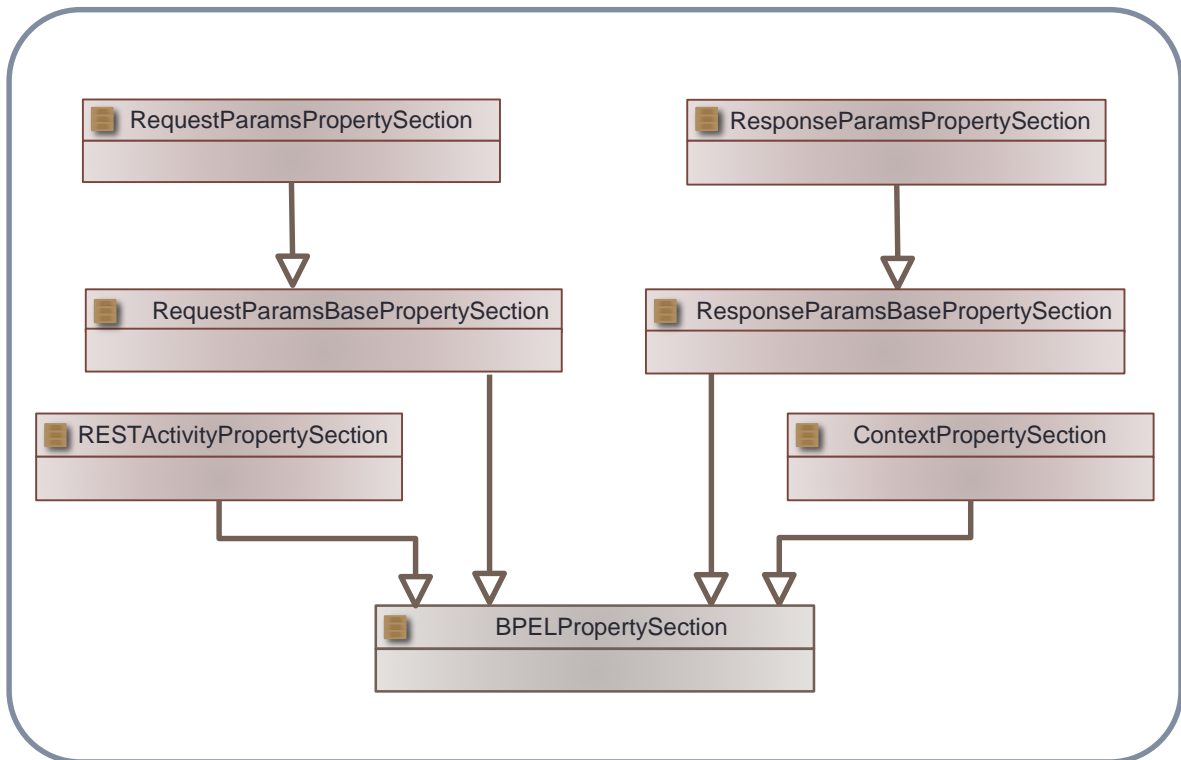


Abbildung 5.5: Vererbungshierarchie der PropertySection-Klassen

Für jeden Reiter der Ansicht „Eigenschaften“ wird eine PropertySection-Klasse verwendet. Die *RESTActivityPropertySection* füllt den Reiter „Details“ mit Steuerelementen für die Attribute „host“ und „path“ aus. Die weiteren PropertySection-Klassen befüllen die Reiter „Context“, „Request Parameters“ und „Response Parameters“ entsprechend dem im Kapitel 4.2 vorgestellten UI Prototyp.

Die Hierarchie der PropertySection-Klassen ist in der Abbildung 5.5 dargestellt. Die Vererbung *Request-* und *ResponseParamsPropertySection* von entsprechenden BasePropertySection ist dadurch bedingt, dass alle Typen der REST-Aktivitäten eine Anzahl der gleichen grundlegenden Eigenschaften haben. Die Steuerelemente für diese Eigenschaften werden in der *RequestParams-* bzw. *ResponseParamsBasePropertySection* erstellt.

Alle PropertySection-Klassen erben von der abstrakten Klasse des BPEL Designers, nämlich *BPELPropertySection*. Die folgenden Methoden werden dabei überschrieben:

- **void createClient(Composite parent):** In dieser Methode werden Steuerelemente mit dem zugehörigen Controller erstellt und in Reiter hinzugefügt, Beobachter registriert und Kommandos, die bei Änderungen ausgeführt werden sollen, definiert.
- **void basicSetInput(EObject newInput):** Als Eingabeparameter bekommt die Methode eine Instanz der Modellklasse, die in plugin.xml für diese PropertySection definiert ist. Die Steuerelemente werden initialisiert und die entsprechenden Controller bekommen Modellobjekte als Input.
- **MultiObjectAdapter[] createAdapters():** Die Methode gibt Adapters zurück, die bei Änderungen im Modell benachrichtigt werden.

5.3.4 Controller-Komponente

Quelle: [12]

Als Controller-Klassen verwendet die *RESTModelingExtension* die Klassen des BPEL Designers **LeafEditPart** und **EditController**. Die *LeafEditPart*-Klasse ist eine der im BPEL Designer verwendeten *EditPart*-Klassen. Im Allgemeinen verbinden die *EditPart*-Klassen Elemente des EMF-Modells mit GEF-Figuren. Eine *EditPart*-Klasse ist zuständig für [24]:

- Erstellung einer grafischen Darstellung, die das Modellobjekt repräsentiert,
- Beobachtung der Modelländerungen und Anpassung der Darstellung,
- Bereitstellen einer Liste der Kind-Elemente des Modells, die grafisch dargestellt werden sollen.

Die *LeafEditPart* unterscheidet sich von anderen *BPELEditParts* in dem, dass sie für die Aktivitäten eingesetzt wird, die keine Kind-Aktivitäten haben. In der *RESTModelingExtension* wird die Klasse *LeafEditPart* verwendet, um Instanzen der von der *RESTActivity* geerbten Klassen mit ihrer Darstellung zu verbinden.

Ähnlich den *EditPart*-Klassen dient die *EditController*-Klasse der Synchronisierung der Elemente des Modells mit ihrer grafischen Darstellung. Dafür beobachtet der *EditController* sowohl das Objekt des Modells als auch das entsprechende grafische Steuerelement.

Die *RESTModelingExtension* erstellt für jedes editierbare Steuerelement eine Instanz der *EditController*-Klasse. Um die benötigte Arbeitsweise des *EditController*s zu gewährleisten, wird bei der Erstellung einer *EditController*-Instanz eine Referenz auf ein *Command Framework* übergeben, das die Ausführung von Kommandos im BPEL-Editor steuert. Außerdem bekommt die erstellte Instanz Referenzen auf das Steuerelement und auf das entsprechende Modellobjekt, die in seinen Feldern *fViewValue* und *fModelValue* gespeichert werden. Nachdem die *EditController*-Instanz sich als Beobachter des Modellobjektes registriert hat, bekommt sie Benachrichtigungen, falls das Modellobjekt geändert wurde, und ändert dementsprechend die grafische Darstellung. Zudem registriert sich die Instanz als Beobachter der verschiedenen Ereignisse des Steuerelementes. Treten diese Ereignisse von Benutzerseite auf, wird das *Command Framework* darüber benachrichtigt, um die Änderungen auf das Modellobjekt zu übertragen.

Auf dem Beispiel des Sequenz-Diagramms, das in der Abbildung 5.6 zu sehen ist, wird die Zusammenarbeit MVC erläutert. Hier wird ein Vorgang gezeigt, der beim Betätigen eines Steuerelementes der Reiter „Context“ angestoßen wird. Allerdings sind einige Schritte übersichtshalber aus dem Diagramm ausgelassen. Nachdem der Benutzer das Editieren des Textfeldes „Cachecontrol“ beendet hat, bekommt der beobachtende *EditController* ein Ereignis vom *Text-Widget*. Der *EditController* prüft, ob nach dem Eintreten dieses Ereignisses weitere Aktionen erforderlich sind, und übergibt die Steuerung an das *Command Framework*. Das *Command Framework* sorgt für die Ausführung eines Kommandos, das den Wert des im *fViewValue* referenzierten Textfeldes ausliest und in das Modellobjekt einsetzt. Bei der Zuweisung des *Cachecontrol*-Wertes dem Feld des *Context*-Objekts wird der *RESTReconciliationHelper* aufgerufen der den Wert des entsprechenden XML-Elements ändert. Demgemäß werden die durch das Betätigen des Steuerelementes entstandenen Änderungen sowohl in das EMF-Modell als auch in die BPEL-Datei übertragen.

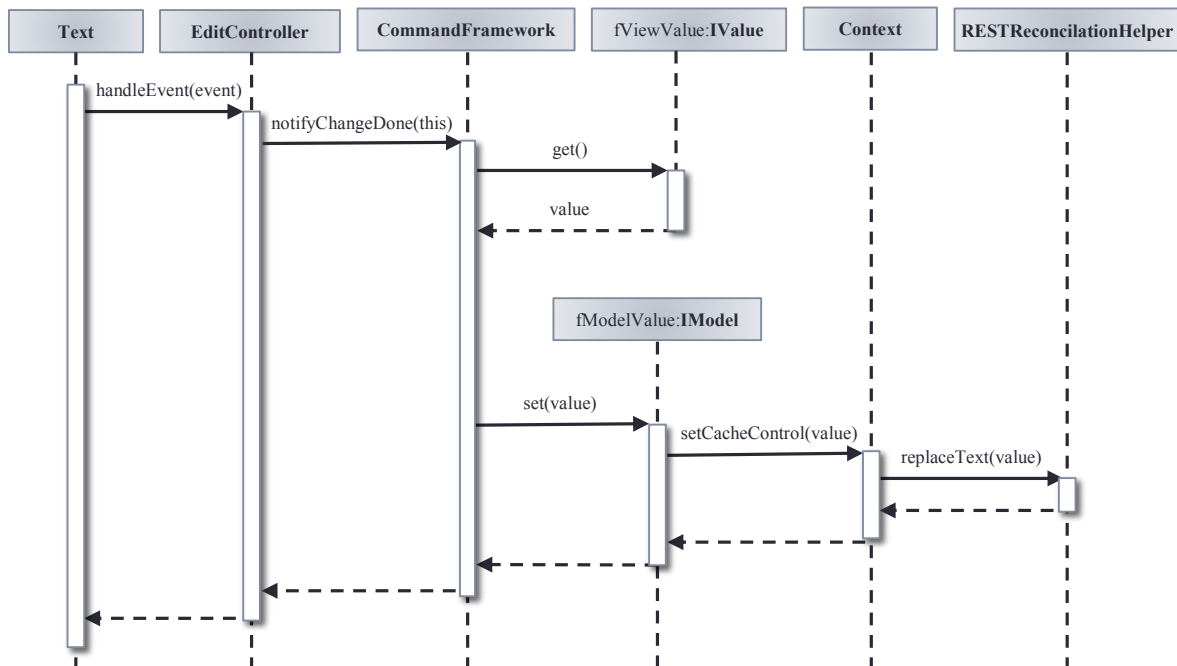


Abbildung 5.6: Zusammenarbeit MVC

5.3.5 Validierung

Die Validierung der Benutzereingabe findet an verschiedenen Stellen statt. Sowohl in der grafischen Ansicht als auch beim Editieren des Quellcodes der BPEL-Datei wird der Benutzer durch Validierung unterstützt. Dementsprechend kann man zwei Gruppen von Aktionen auszeichnen.

Zu der ersten Gruppe gehören eingebaute Kontrollaktionen beim Schließen des Dialogfensters. Diese Aktionen sorgen für eine Validierung der Eingabe, bevor das Dialogfenster geschlossen wird. Bei eventuellen Fehlern bekommt der Benutzer eine Benachrichtigung, dazu wird die **MessageDialog**-Klasse verwendet. Außerdem sind in diese Gruppe Aktionen eingeschlossen, die für eine Beobachtung der Steuerelemente zuständig sind. Zu den Steuerelementen werden **SelectionListener** bzw. **KeyListener** hinzugefügt, die die Eingabe des Benutzers beobachten. Falls der Benutzer mit einem Auswahlkästchen ein verpflichtendes Textfeld zur Eingabe freischaltet bzw. die Eingabe eines der verpflichtenden Textfelder löscht, wird dieses Textfeld sofort markiert.

Zu der zweiten Gruppe gehören die Aktionen, die für die Validierung des Inhalts der BPEL-Datei sorgen. Dafür wird eine Erweiterung der Extension-Points des Validierungs-Plug-ins des BPEL Designers verwendet. Hier wird ein **RESTValidatorPlugin** angeschlossen, wie es im Unterkapitel 5.2 beschrieben wurde.

Die wichtigsten Klassen des **RESTValidatorPlugin** sind:

- **RESTFactory**: Eine Fabrik-Klasse, die Interface **IFactory<Validator>** implementiert, wird für die Erstellung der Objekte der Validator-Klassen verwendet.

- **PostValidator, AuthenticationValidator** usw.: Für jedes XML-Element, das validiert werden soll, existiert eine Validator-Klasse, die das XML-Element nach der in der BPEL4REST spezifizierten Syntax überprüfen kann. Die Methoden der Validator-Klassen, die die Regeln enthalten, werden mit „@ARule“ annotiert, damit der BPEL-Validator sie als solche erkennen kann.

Wie in der Abbildung 5.7 zu sehen ist, erben alle Validator-Klassen von der Klasse des BPEL Designers *CValidator*. Infolge dessen werden Regeln zur Überprüfung der Vater-Kind-Beziehungen für alle REST-XML-Elemente angewendet. Um die verpflichtenden Kind- und erlaubten Vater-Elemente in einer Validator-Klasse festzulegen, wird die Methode **void checkChildren()** bzw. **IFilter<INode> parentNodeNames()** überschrieben. Regeln, die überprüfen, ob der Inhalt der XML-Elemente und der Attribute spezifikationskonform ist, werden zusätzlich definiert.

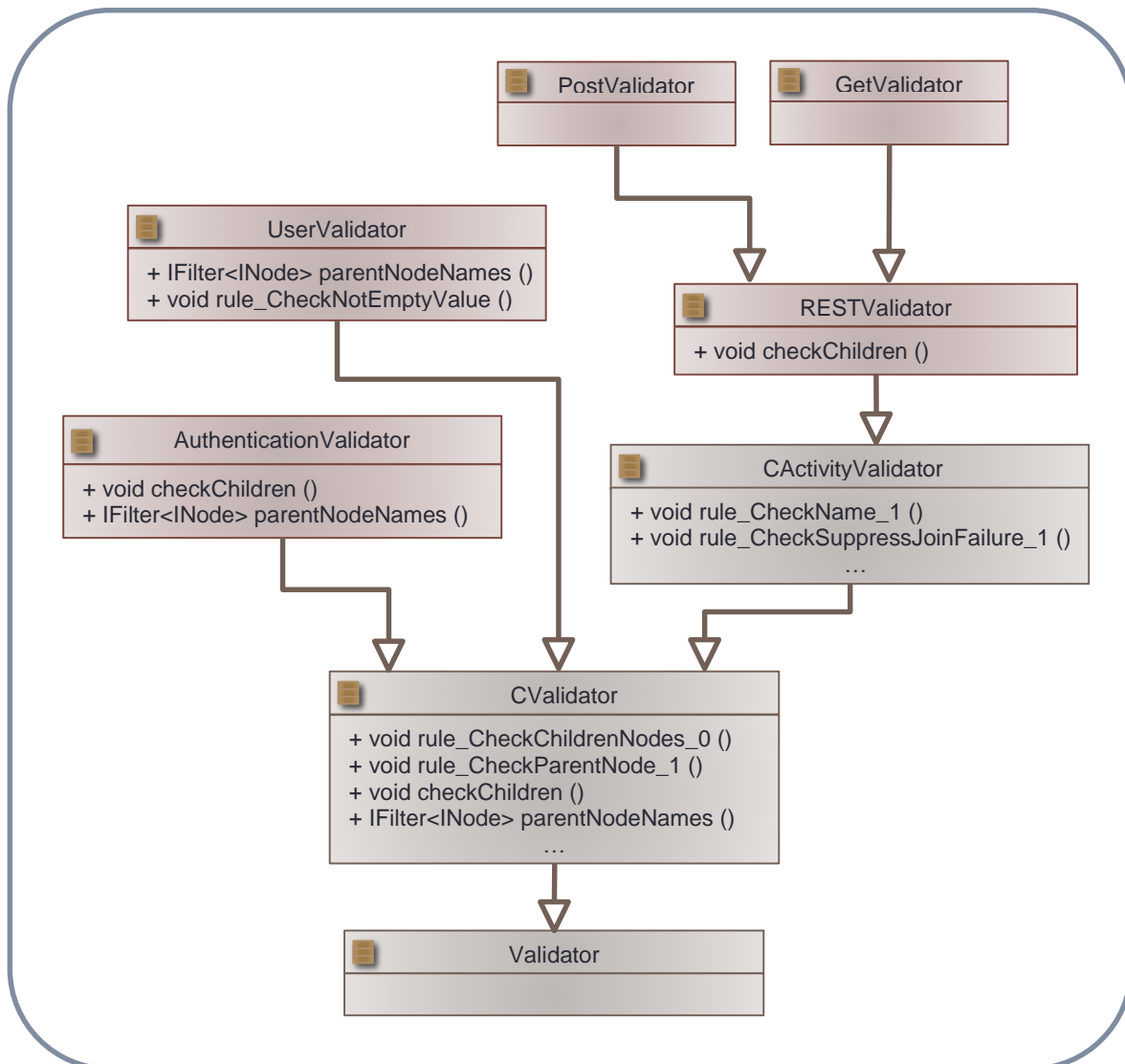


Abbildung 5.7: Ausschnitt aus dem Vererbungsdiagramm der Validator-Klassen

5.4 Einschränkungen

Die Implementierung der RESTModelingExtension wurde unter Berücksichtigung einer Einschränkung durchgeführt: Die RESTModelingExtension wird als eine Erweiterung des **SimTech BPEL Designers** betrachtet und kann nicht ohne weiteren Änderungen mit dem Eclipse BPEL Designer verwendet werden.

Der Grund dafür ist die Tatsache, dass das EMF-Modell der RESTModelingExtension unter der Verwendung des EMF-Modells des SimTech BPEL Designers aufgebaut wurde. Dementsprechend erben die Klassen der RESTModelingExtension von den Klassen des SimTech BPEL Designers, die im Vergleich zum Eclipse BPEL Designers zusätzliche Attribute haben.

Allerdings verwendet die RESTModelingExtension keine spezifischen Elemente des SimTech BPEL Designers. Folglich ist es nicht kompliziert, die Erweiterung an den Eclipse BPEL Designer anzupassen, dafür muss lediglich das Modell ersetzt werden.

6 Zusammenfassung und Ausblick

Die RESTModelingExtension ist eine Erweiterung des BPEL Designers, die in dieser Arbeit konzipiert und prototypisch implementiert wurde. Die Erweiterung fügt zusätzliche grafische Elemente in die Palette des BPEL Designers ein, um den Modellierer bei der Orchestrierung von REST Services zu unterstützen.

Im Laufe der Arbeit wurden zuerst die verwendeten Begriffe wie REST und BPEL erläutert, die zum Verständnis dieser Arbeit wichtig sind. Außerdem wurden die bestehenden Werkzeuge zur Modellierung der BPEL-Prozesse beschrieben.

Anschließend wurde die BPEL4REST-Erweiterung der BPEL vorgestellt, die dem Benutzer Möglichkeiten anbietet nicht nur WSDL-basierte Services, sondern auch REST Services mit BPEL zu orchestrieren. Die Erweiterung besteht im Definieren der fünf neuen Kind-Elemente für das BPEL-Element `<extensionActivity>`, die jeweils ein HTTP-Verb repräsentieren. Diese Erweiterung wurde analysiert und anhand kleiner Änderungen angepasst um die Struktur der eingeführten Elemente zweckmäßig zu gestalten.

Im Kapitel „Entwurf“ 4 wurden verschiedene Ansätze zur Darstellung REST-Aktivitäten besprochen und ein am meisten sinnvoller Ansatz wurde für die weitere Ausarbeitung ausgewählt. Weiterhin wurden die Prototypen der Benutzerschnittstelle, die die Platzierung der Steuerelemente auf den Reitern „Eigenschaften“ demonstrieren, vorgeschlagen und erklärt.

Im fünften Kapitel wurde die Implementierung der Erweiterung dargelegt. Angefangen mit einer Vorführung der verwendeten Technologien für das Implementieren und Testen wurden im weiteren Verlauf einige Besonderheiten der Plug-in-Entwicklung erläutert. Anschließend die Architektur der Erweiterung vorgestellt und die einzelnen Komponenten beschrieben. Am Ende des Kapitels wurden die Einschränkungen, die während der Implementierung angenommen wurden, aufgelistet.

6.1 Ausblick

Die RESTModelingExtension wurde nur prototypisch implementiert, aus diesem Grund sind weitere Verbesserungen der Erweiterung möglich. Einige der Möglichkeiten sind im Folgenden aufgelistet.

Die Validierung des Quelltextes von BPEL-Datei beruht auf der Spezifikation der BPEL4REST. Zusätzlich wäre es möglich, die Validierungsgrundlage zu erweitern und den Inhalt der Elemente und Attribute, die laut BPEL4REST-Spezifikation einen HTTP-konformen String-Wert annehmen sollen, auf das Entsprechen des HTTP-Standards zu prüfen.

Aus den gleichen Überlegungen wurde für das Editieren mancher String-Elemente und Attribute ein Textfeld als Steuerelement ausgewählt. Als eine Weiterentwicklung wäre denkbar, die vorhandenen Textfelder mit Auswahllisten oder Auswahldialogen zu ersetzen, um dem Benutzer, der nur wenige HTTP-Kenntnisse besitzt, das Ausfüllen der HTTP-Headers zu erleichtern.

Sowohl die BPEL4REST als auch RESTModelingExtension basieren auf ihren eigenen Datenmodellen, die momentan nicht miteinander verbunden sind. Dies führt dazu, dass alle Änderungen an dem Datenmodell von BPEL4REST das Anpassen des Datenmodells von RESTModelingExtension nach sich ziehen. Es wäre gut, eine gemeinsame Basis für beide Datenmodelle zu entwickeln und zu verwenden, damit die möglichen Änderungen oder Erweiterungen der XML-Spezifikation der BPEL4REST automatisch auf das Datenmodell der RESTModelingExtension übertragen werden.

Literaturverzeichnis

- [1] F. Haupt, M. Fischer, D. Karastoyanova, F. Leymann und K. Vukojevic-Haupt, *Service Composition for REST*, 2014.
- [2] R. T. Fielding, „Architectural Styles and the Design of Network-based Software Architectures“, 2000. [Online]. Available: <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.
- [3] N. Freed und N. Borenstein, „Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types“, 1996. [Online]. Available: <http://www.ietf.org/rfc/rfc2046.txt>.
- [4] R. T. Fielding, J. Gettys, J. Mogul, H. Frystyk, P. L. L. Masinter und T. Berners-Lee, „Hypertext transfer protocol–HTTP/1.1.“, 1999. [Online]. Available: <http://www.ietf.org/rfc/rfc2616.txt>.
- [5] L. Masinter, T. Berners-Lee und R. Fielding, „Uniform resource identifier (URI): Generic syntax“, 2005. [Online]. Available: <http://www.ietf.org/rfc/rfc3986.txt>.
- [6] F. Haupt, D. Karastoyanova, F. Leymann und B. Schroth, „A model-driven approach for REST compliant services“, in *Proceedings of the IEEE International Conference on Web Services, ICWS*, 2014.
- [7] „SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)“, 2007. [Online]. Available: <http://www.w3.org/TR/2007/REC-soap12-part1-20070427/>.
- [8] S. Tilkov, *REST und HTTP Einsatz der Architektur des Web für Integrationsszenarien*, dpunkt.verlag, 2011.
- [9] „OASIS Web Services Business Process Execution Language Version 2.0“, 2007. [Online]. Available: <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>.
- [10] „Apache ODE“, [Online]. Available: <http://ode.apache.org/>.
- [11] „Oracle BPEL Process Manager“, 2009. [Online]. Available: <http://www.oracle.com/technetwork/middleware/bpel/overview/ds-bpel-11gr1-1-134826.pdf>.
- [12] „BPEL Designer Project“, [Online]. Available: <http://www.eclipse.org/bpel/>.
- [13] M. Hahn, R. El-Hussein und B. Schroth, *Benutzerhandbuch SimTech-Prototyp*, 2013.

- [14] M. Fischer, *RESTful BPEL - Erweiterung von BPEL zur Orchestrierung von RESTful Web Services*, 2013.
- [15] „Eclipse,“ [Online]. Available: <http://www.eclipse.org>.
- [16] „Eclipse EMF,“ [Online]. Available: <http://www.eclipse.org/modeling/emf/>.
- [17] „Eclipse GEF,“ [Online]. Available: <https://projects.eclipse.org/projects/tools.gef>.
- [18] M. Hahn und K. Vukojevic, *Installationsanleitung SimTech Prototyp*, 2013.
- [19] „Eclipse EMF Compare,“ [Online]. Available: <https://projects.eclipse.org/projects/modeling.emf.compare>.
- [20] „Eclipse GMF Runtime,“ [Online]. Available: <https://projects.eclipse.org/projects/modeling.gmp.gmf-runtime>.
- [21] „Apache Tomcat,“ [Online]. Available: <http://tomcat.apache.org/>.
- [22] „Apache ActiveMQ,“ [Online]. Available: <http://activemq.apache.org/>.
- [23] „Notes on the Eclipse Plug-in Architecture,“ 2003. [Online]. Available: http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html.
- [24] D. Rubel, J. Wren und E. Clayberg, *The Eclipse Graphical Editing Framework (GEF)*, Addison-Wesley, 2011.

Alle Links zuletzt geprüft am 11.11.2014

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift