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

Diplomarbeit Nr. 3079

Entwicklung eines Frameworks zur Verwaltung von abstrakten Sichten auf BPEL Prozesse

Jiayang Cai

Studiengang:	Informatik
Prüfer:	Prof. Dr. Frank Leymann
Betreuer:	Dipl.-Inf. David Schumm
begonnen am:	27. Juli 2010
beendet am:	26. Januar 2011
CR-Klassifikation:	D.2.2, H.4.1, H.5.2

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Listings

Danksagung

An dieser Stelle möchte ich mich bei meinem Betreuer Dipl.-Inf. David Schumm bedanken, der mich während meiner Studienarbeit und Diplomarbeit betreut hat und mich bei der Erstellung, Formulierung, Korrektur dieser Arbeiten immer Rede und Antwort stand. Ich möchte mich bei den Kommilitonen und den Lernpartnern bedanken, die mir in meinen Informatikstudium an der Uni Stuttgart geholfen haben. Weiterhin möchte ich mich bei Professor Dr. Frank Leymann und Jun.-Prof. Dr.-Ing. Dimka Karastoyanova für die bedeutungsvollen Veranstaltungen bedanken, die mich in die Themengebiete Softwarearchitektur und Workflow-Management eingeführt haben.

Besonders bedanken möchte ich mich aber bei meinen Eltern, ohne die dieses Studium in Deutschland nie möglich gewesen wäre. Bei meiner Bruder möchte ich mich für die ständige Ermunterung und Unterstützung bedanken. Zudem möchte ich bei meiner Freundin danken, die mich immer unterstützt und mir den Rücken gestärkt hat.

1. Einleitung

Seit Anfang der 90er Jahre hat sich Geschäftsprozessmanagement als fester Bestandteil der Anwendungssystem- und Organisationsgestaltung in der Praxis etabliert [GSVR94]. Es wird in der Wissenschaft und Industrie ständig erforscht, einen agilen und effizienten Geschäftsprozess in den Unternehmen zu gewährleisten und zu optimieren. Ein besser passender Geschäftsprozess bringt den Unternehmen langfristig mehr Erfolg und strategische Vorteile bzw. die größte Konkurrenzkompetenz in der Branche und beste Geschäftsorientierung für ein nachhaltiges Unternehmenswachstum. Das Geschäftsprozessmanagement umfasst alle Tätigkeiten, die sich auf den Geschäftsprozess beziehen, sowie das Identifizieren, Modellieren, Dokumentieren, Durchführen, Überwachen, Analysieren und die kontinuierliche Verbesserung von Geschäftsprozessen. Der Geschäftsprozess ist heutzutage wegen den vielfältigen Kundenanforderungen und ständige Geschäftsverwandlungen bzw. die Neuproduktentwicklung sehr kompliziert. In dieser Arbeit diskutieren wir nicht darüber sowie wie man das Geschäftsprozessmanagement in den Unternehmen besser durchführen und welche Softwarewerkzeug und Methode sollen die Prozessexperte anwenden. Wir betrachten die Geschäftsprozess-Transformation, die ein nützliches Instrument für die Komplexitätsreduzierung beim Prozess Engineering ist.

1.1. Motivation

Wie stellt man es an einen Geschäftsprozess in einem Großkonzern z.B in der Öl- und Gasindustrie zu analysieren und zu verbessern? In solch einer Branche ist der Geschäftsprozess ein Kernfaktor für den Unternehmenserfolg. Der Geschäftsprozess greift auf die verschiedenen Standorten auf verschiedenen Kontinenten und auf Arbeitsgruppen bzw. Projektmitarbeitern in unterschiedenen Funktionssegmenten zu. Die Komplexität des Geschäftsprozess ist streng abhängig von dem Grad der Detaillierung und der Verflechtung von den Subprozessen. Da eine Komplexitätsreduzierung besonders gefordert ist, werden in der Industrie viele geschäftliche Ansätze dafür erstellt und veröffentlicht, die nach den entsprechenden Branchenmerkmalen und Geschäftsverhalten z.B in Bank und Automobile spezifiziert sind. In der Arbeit [SLS10] wurde eine grundlegende Methode »Process View« für die Komplexitätsreduzierung und Prozesstransformation herausgegeben, mit der die ungewollten Details eines Prozess behoben und der Prozess auf das gewünschten Anwendungsziel abstrahieren und transformieren können. Prozess View Transformation ist eine XML-basierte Methode für die spezifizierte Transformation einer durch eine Prozessbeschreibungssprache z.B WS-BPEL [AAA+07] erstellten Geschäftsprozess. Er generiert eine definierte Sicht auf den

Gesamtgeschäftsprozess für die Benutzer. Der in den Arbeiten [Cai10][Str09] entwickelte Prototyp und Implementierungskonzept für Prozess View Transformation zeigt eine stärke Abstraktionsfähigkeit an. Es wird in dieser Diplomarbeit dieses Modell vervollständigt und gegebenenfalls weiterentwickelt.

1.2. Zielsetzung

Um eine Prozess View Transformation zu erstellen benötigt man bisher die Kenntnisse der Rules-Sprache und Verarbeitungsprinzipien, eine Konsole-basierte Jar-Anwendung, wie in die Arbeite [Cai10][Str09] bietet wenige komfortable Benutzerfreundlichkeit und bewirkt die hohe Komplexität der Prozessabstraktion. Eine Webanwendung für die flexible Prozesstransformation wird in dieser Diplomarbeit entwickelt. Es wird dafür gefordert, eine Webservices-basierte Verwaltungsplattform für die vereinfachte Konstruktion der Prozessabstraktion zu realisieren, ein interaktiver Rules-Designer wird für ein anschauliches Editieren von Rules Dokument entwickelt, ohne eine manuelle Erstellung zu benötigen. Dieses entworfene Framework richtet sich an eine serviceorientierter Architektur und stützt eine flexible Erstellung von Prozess View Transformation Vorlagen, die selbst als Webservices für die definierte Prozesstransformation und als Anwendungsziel zur Verfügung gestellt werden können.

1.3. Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich wie folgt in fünf Kapiteln:

Kapitel 1 - Einleitung In diesem Kapitel wird in das Themengebiet Business Prozess View und die Aufgabestellung der Diplomarbeit eingeführt, die Motivation und Zielsetzung werden dabei erklärt.

Kapitel 2 - Grundlagen und Technologien Die grundlegende Theorien in Webservices, Workflows und die technische Ansätze bzw. angewandten Frameworks in der Webanwendungsentwicklung werden in diesem Kapitel vorgestellt, die für die technische Implementierung der praktische Aufgabe der Arbeit notwendig sind.

Kapitel 3 - Konzept und Entwurf In diesem Kapitel werden die Konzeption und der erstellte Entwurf für die Entwicklung der Plattform bzw. des Frameworks zur Verwaltung von abstrakten Sichten auf BPEL-Prozess erläutert.

Kapitel 4 - Implementierung Die praktische Implementierung und einige Anwendungsfälle der Verwaltungsplattform für abstrakte Sichten werden in diesem Kapitel in Details erläutert.

Kapitel 5 - Zusammenfassung und Ausblick In dem letzten Kapitel der Arbeit wird die Schlussfolgerung dieser Diplomarbeit kurz zusammengefasst und einen Ausblick bzw. die mögliche Verwandlung vom Thema »Business Process View« in den künftigen Anwendungsaspekten gegeben.

2. Grundlagen und Technologien

In diesem Kapitel werden die wichtigen und grundlegenden Theorien in Web Services und die eingesetzte Technologien aus dem JEE-Umfeld vermittelt, auf denen diese Diplomarbeit basiert. Die fundamentale Definition und Prinzipien in den Themengebiet von Web Services und entsprechende Standardisierungen werden zuerst erläutert. Die theoretischen Grundlagen der Business Process Execution Language aus [OAS07] und des Business Process View aus [Str09] [Cai10] werden kurz erklärt. Die Technologien und Entwurfsmuster im JEE-Umfeld werden vorgestellt, die in der praktischen Arbeit bei der Realisierung der Webanwendung und der Integration des Frameworks angewendet wurde.

2.1. Web Services

Web Services werden heutzutage als die beste und effizienteste Implementierungsmethode von Service-orientierten Architekturen betrachtet und weiterhin vom World Wide Web Consortium (W3C) als eine informationstechnische Standardlösung gepflegt. Eine vollständige Definition von Web Services wird in [BHM+04] von W3C für die künftige bessere Zusammenarbeit und Weiterentwicklung wie folgt vereinbart:

> *A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*

Web Services sind nicht eine allein stehende Technologie, es ist ein Bündnis von mehreren Spezifikationen und Standards für die verschiedenen klassifizierten Funktionsschichten. Die Web Services Gemeinde [1] entwickelt zusammen mit den Experten und führenden industriellen Partnern in der IT-Branche die Technologien, um die Kompetenz und Anwendbarkeit von Web Services zu verbreiten und zu verstärken. Webservice strebt die beste Zusammenarbeitsfähigkeit und höchste Plattform-Unabhängigkeit in der realen Anwendungsentwicklung und Systemintegration an, eine schematische Illustration von der entwickelten Spezifikationen des Webservice-Stack können in [WCL+05, Kapitel 3.1] gefunden werden, die die verschiedenen konkreten Funktionen in der serviceorientierten Geschäftsanwendungsentwicklung realisieren und erweitern können. Solche ständig betreuten Spezifikationen definieren die

[1] http://www.w3.org/2002/ws/

Nachrichtenformate, Transportprotokolle und Beschreibung der vereinbarten Servicequalität usw., um eine beste Interoperationalität in der Web Services Welt zu realisieren.

2.1.1. WSDL

Die Web Services Description Language (WSDL) 1.1 [2] ist eine standardisierte Spezifikation bzw. eine XML-basierte Metasprache für eine operative Beschreibung von Webservice. WSDL spielt eine entscheide Rolle bei der praktischen Implementierung von Web Services Konzepten und des SOA-Paradigma. In der Softwareentwicklung und in der unternehmensweiten Anwendungsintegration sind die gemeinsame Vereinbarung und Einhaltung auf den durch WSDL definierte Servicestand und spezifizierte Qualitätsbeschreibung bei der Analyse- und Designphase sehr sinnvoll, weil es eine flexible, dynamische, lose-gekoppelte Anbindung für die Anwendungen in eine interaktive und sich ständig veränderte Umgebung schafft.

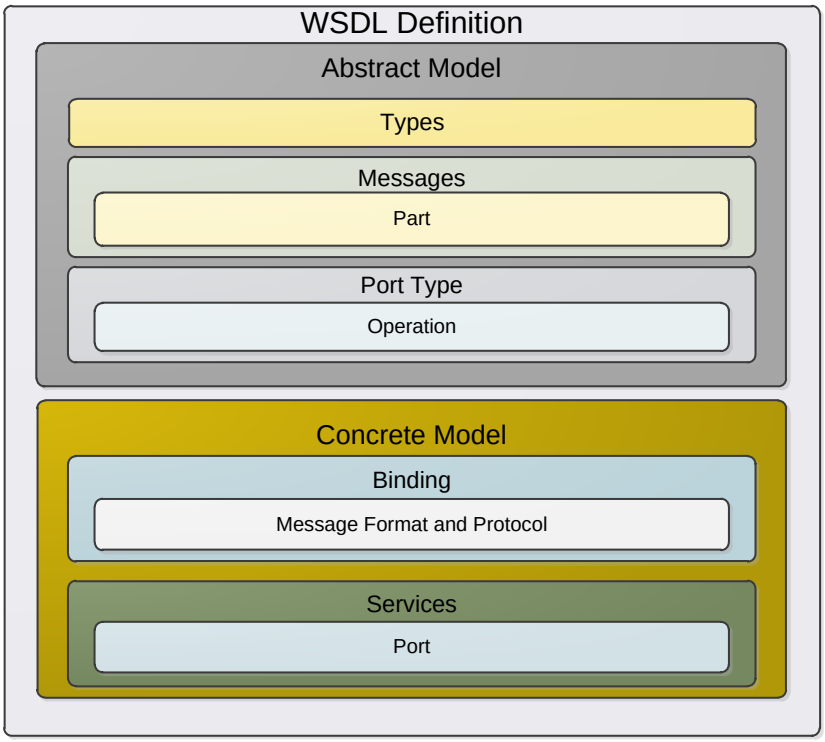

Abbildung 2.1.: WSDL Definition und Komponenten aus [WCL+05]

Ein WSDL Dokument beschreibt einen veröffentlichten Service für die Serviceanwender, mit den erwünschten Nachrichtenformaten, die verfügbaren Operationen zu aufrufen, die Lokalisierung des Services und die Transportprotokolle für die Interaktion werden festgesetzt.

[2] http://www.w3.org/TR/wsdl

Die Serviceanwender können wegen dieser operativen Beschreibung den Service effizient und problemlos benutzen. Eine von Anwendern erstelltes WSDL Dokument ist für die Serviceanbieter einen Serviceauftrag. Die Anwender beschreiben detaillierte die eignen Anforderung an die Funktion und Kommunikation des Service bzw. Dienstleistung in einem WSDL Dokument. Das WSDL Dokument in Web Services ist deswegen eine neutrale und funktionale Beschreibungsvermittler in der heterogen Infrastruktur der reale Welt.

Ein anschauliches Blockdiagramm in der Abbildung 2.1 stellt die WSDL Dokumentdefinition und die spezifische Konzeption dar. Ein WSDL Dokument beinhaltet typischerweise zwei definierte Beschreibungen wie in der obige Abbildung, einen abstrakte Beschreibungsteil und einen exakten Beschreibungsteil. In dem abstrakten Beschreibungsteil wird eine allgemeine Definition von Web Services gegeben, es stellt die Funktionalität und Anwendbarkeit von Web Services durch die assoziierte Operationen und die entsprechend erwünschte oder generierte Nachrichten dar, ohne eine detaillierte Angabe für die Aufrufmethode bzw. das verfügbare Transportprotokoll und den genau Endpunkt zu beschreiben. In den dreidimensionalen Beschreibungskoordinaten (Was-Wie-Wo) bedeckt die abstrakte Beschreibungsteil die Was-Dimension. Im exakten Beschreibungsteil des WSDL Dokument werden die genauen Hinweise für das Aufrufen und die Implementierung von Web Services gegeben. Die Wie-Dimension wird durch die <binding> Element in WSDL beschrieben, die Wo-Dimension in den Koordinaten wird durch das <service> Element im exakten Beschreibungsteil dargestellt. Eine aufgelistete Kurzfassung der Kernelemente in der WSDL Spezifikation Version 1.1 wird im folgenden Abschnitten erstellt.

Types Im XML Element <types> werden die Datentypen und Datenstruktur der assoziierten Parts bzw. Bausteinen in den ausgetauschte Nachrichtenformaten beschrieben. Die Anwendungsbedeutung und Syntax von <types> ist identisch mit dem <schema> Element im XML Schema (XSD) [3].

Message Das <message> Element beinhaltet eine oder mehrere logische Parts bzw. Nachrichtenbausteine. Jeder Part des Nachrichtenelement wird mit dem entsprechenden Typenelement in einem Typendefinitionssystem assoziiert. Dadurch beschreibt es den logisch abstrakten Kontext der ausgetauschten Nachrichten. Wenn eine Nachricht durch RPC (Remote Procedure Call) Transportprotokoll ausgetauscht wird, sind die Parts im Nachrichtenelement sind die bedeutungsvollen Parametern für die Eingabe und Ausgabe des Operationsaufruf.

Operation Im XML Element <operation> in WSDL wird eine abstrakte Kurzbeschreibung der Operationen oder der verfügbaren Implementierungsaktionen aus den Web Services gegeben. Jede Operation im WSDL Dokument hat einen eigenen eindeutigen Namen und enthält die spezifizierten Eingangs- und Ausgangsnachrichten bzw. Fehlermeldungen bei einem fehlgeschlagene Operationsaufruf. Es werden in der Implementierung vier Arten von Operation in den Endpunkt realisiert: Der Endpunkt bekommt eine Nachricht und gibt keine Rückmeldung, der Endpunkt generiert eine entsprechende Antwort für die ankommende Nachricht, der Endpunkt sendet eine ursprünglich Anfrage und wartet auf die Antwort oder signalisiert nur die Verbreitung einen Bescheid.

[3] http://www.w3.org/TR/xmlschema-0/

Port Type Ein <portType> Element ist eine Menge von einer oder mehreren <operation> Elementen, die in einem Webservice implementiert sind und in einer bestimmten klassifizierten Sorte angeliefert werden. In einem WSDL Dokument können mehrere <portType>Elementen spezifiziert werden.

Binding Im <binding> Element werden eine aufgeklärte Beschreibung von den Nachrichtenformaten eines Operationsaufruf eines bestimmten <portType> Elements und das entsprechende detaillierte Transportprotokoll definiert. In einem <binding> Element muss genau ein einziges Transportprotokoll spezifiziert werden.

Port Ein <port> Element beschreibt einen Endpunkt für ein bestimmtes <binding> Element in WSDL. Es wird darin eine Netzwerkadresse z.B <soap:address> für SOAP-Binding oder Objektzugang für den Zugriff und Datenaustausch angegeben. Innen ist ein <port> Element welches nur eine Zugriffsadresse spezifiziert.

Service In einem <service> Element in WSDL werden alle <port> Elementen zusammen gruppiert. Es erfasst alle verfügbaren Zugriffsadressen und die spezifizierten aufrufbaren Endpunkte in Web Services.

Die in dieser Arbeit angewendete WSDL Spezifikation ist die Version 1.1, eine weiterentwickelte WSDL von Version 2.0 wird in dieser Arbeit nicht diskutiert. Die neuen Verwandlungen und Komponenten-Modelle der neusten Version können in den entsprechende W3C Weblinks [4] gefunden werden.

2.1.2. SOAP

SOAP steht jetzt ab den Version 1.2 [5] nicht mehr für eine ursprünglich Abkürzung von »Simple Object Access Protocol«, sondern ist der Name des fundamentalen Nachrichten-Framework für Web Services. Die spezifiziert das XML-basierten Format und das Verarbeitungsmodell der abgeschickte Nachrichten zum Kommunizieren in Web Services. Eine SOAP-Nachricht ist ein wie eine Versandtasche konstruiertes Strukturformat, die neben dem Transportieren von relevante Daten durch mehrere besondere Funktionen wie z.B die Zielnavigation und die Servicequalitätsbeschreibung erweiterbar ist. Durch den Austauschen von SOAP-Nachrichten zwischen Service-Anbieter, Service-Anwender und den Vermittlern werden Web Services in einem lose gekoppelte Verfahren in verschiedene Implementierungsplattformen und Netzwerksprotokollen für den Transport aufgerufen und ausgeführt. SOAP ist dabei eine der grundlegenden Spezifikationen im Web Services Stack.

[4] http://www.w3.org/TR/wsdl20/
[5] http://www.w3.org/TR/soap12-part0/

2.2. Business Process Execution Language

Web Services-Business Process Execution Language kurz als WS-BPEL [6] ist eine von OASIS (Organization for the Advancement of Structured Information Standards) [7] standardisierte Beschreibungssprache für die Modellierung, Simulation und das Analysieren von den auf Web Services basierten Geschäftsprozessen. WS-BEPL spezifiziert die erweiterbare XML-basierte Aufbaustruktur und geschäftliche Interaktionsprotokolle für die Erstellung von ausführbaren und von abstrakten Geschäftsprozessen. Die Web Services und deren Funktionalitäten werden importiert oder exportiert durch die Interfaces in den BPEL-Prozessen und den entsprechenden Transportprotokollen. WS-BPEL Spezifikation steht aber nicht allein, es ist verbunden mit weiteren XML-basierte Spezifikationen wie WSDL 1.1, XML Schema 1.0, XPath 1.0, XSLT 1.0 and Infoset. Eine genaue technische Spezifikation von WS-BEPL Version 2.0 wurde in Dokumentation [AAA+07] detailliert vorgestellt. Für eine ausführliche Erklärung von WS-BPEL und den neu entwickelten Merkmalen können die weiteren Ressourcen und Dokumentationen in den OASIS BPEL technische Committee Webpräsenz [8] gefunden werden.

2.2.1. Orchestration

Orchestration steht für einen ausführbaren Geschäftsprozess durch die Komposition von Web Services. In den Geschäftsprozessen werden die interne und externe Web Services aufgerufen. Eine Orchestration beschreibt die Interaktionen zwischen den Web Services in einem Geschäftsprozess auf der Nachrichtenebene, den Durchläufen und die zugeordnete Ausführung der Interaktionen werden ebenfalls darin definiert. Die Interaktionen zwischen mehreren Applikationen und Plattformen bilden einen langlaufenden und transaktionalen Geschäftsprozess ab, solche eine Orchestration wird aber durch einzelne Teilnehmer verglichen mit der »Choreography« kontrolliert und geschlossen. Eine Choreography besteht zwischen den Teilnehmern in einem Geschäftsprozess. Jeder Teilnehmer kommuniziert mit dem Prozess entsprechend seiner Rolle, die er im ganzen Geschäftsprozess dabei spielt und nach der er behandelt. Der ganze Geschäftsprozess resultiert durch das Zusammenspiel und die Interaktionen von alle Teilnehmern. Eine standardisierte Spezifikation von Choreogrphy [9] können in W3C unter dem Name »Web Services Choreography Description Language« gefunden werden. Eine wesentlicher Unterschied einer Orchestration ist, dass sie einen Prozessflow zwischen Web Services und der interne geschäftliche Logiken darstellt und zu einem einzigen Hauptteilnehmer gehört. In dieser Arbeit wird nur die Aspekte in Orchestration und deren Implementierungssprache WS-BEPL betrachtet und verwendet. Für eine ausführliche Erklärung zwischen den Orchestration und Choreography können die Dokumenten [AAA+07], [KBRL05] genutzt wenden.

[6] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html
[7] http://www.oasis-open.org/home/index.php
[8] http://www.oasis-open.org/committees/wsbpel/
[9] http://www.w3.org/TR/ws-cdl-10/

2.2.2. Workflow Engine

Eine Workflow Engine ist die Laufzeit-Komponente in einem Workflow-Management System. Eine Worklow Engine steuert, ausführt und überwachte die Instanzen des eingesetzten Geschäftsprozessen zusammen mit der verbundene Datenbank aus. Eine Workflow Engine wird in der Praxis für den Aufbau einer Plattform genutzt, um die in WS-BPEL modellierte Servicesequenz für die Prozesstransformation im Server ausführen und selber als Web Services aufrufbar sein zu können. Es werden in den folgenden Abschnitten die in dieser Arbeit angewendete Workflow Engine Apache ODE und seine Architektur erklärt.

Apache ODE

Apache ODE (Orchestration Director Engine) [10] ist eine auf Java implementierte frei lizenzierte Workflow-Engine für die Ausführung der durch standardisierte WS-BPEL erstellten Geschäftsprozessen. ODE verwaltet die definierte Geschäftslogik im Prozess und die Kommunikation zwischen den eingesetzten BPEL-Prozessen und Web Services. Er schickt die SOAP-Nachrichten ab und empfängt die entsprechende SOAP-Nachricht-Antwort. Die in den BPEL-Prozess spezifizierte Operationsaufrufen, Datenmanipulation und Fehlerbehandlung werden in Apache ODE implementiert und überwacht. Die aus Web Services kompositionierte transaktionellen kurzlebigen BPEL-Prozess oder langlebige Prozesse für eine lang laufend Geschäftsapplikation sind ebenfalls in ODE implementierbar.

Neben der Standardfunktion für die Ausführung von BPEL-Prozessen werden die weiteren Merkmale angeliefert, die besonders für die Entwicklung und Management von eingesetzten BPEL-Prozessen geeignet sind, z.B die vollständige Management-API wird als Webservice angeboten. Diese API fragt den ODE Server über den Zustand des BPEL-Prozesses und der Instanzen ab. Es wird in der Arbeit diese Anwendung benutzt, um den aktuell Zustand der aus WS-BPEL erstellten Servicesequenz ebenfalls als BPEL-Prozess zu befragen. Es ist in der Tat vor den Prozessaufrufen wichtig zu wissen, ob die Prozesse im ODE Server bereits erfolgreich eingesetzt, fehlerfrei kompiliert und zur Verfügung gestellt wurde.

In Abbildung 2.2 wird eine schematische Darstellung der Apache ODE Architektur und die entsprechenden kommunizierenden Artefakte bzw. Akteuren gegeben. Diese ist prinzipiell entlehnt aus dem Weblink [11] angezeigt. Die verschiedenen teilnehmenden lose-gekoppelte Systemmodule in ODE und die Interaktion zwischen die Kernkomponenten werden im Folgenden erklärt.

[10] http://ode.apache.org/
[11] http://ode.apache.org/architectural-overview.html

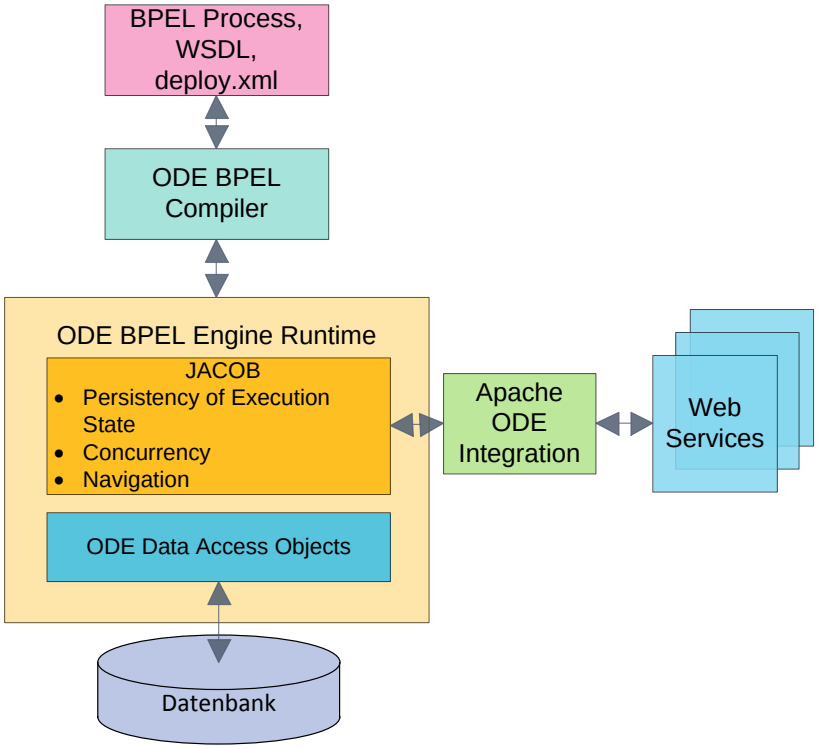

Abbildung 2.2.: Apache ODE Architektur und Komponenten

Die Kernkomponenten der Apache ODE Architektur sind die drei lose gekoppelte Module in der obigen Abbildung dargestellte Objekten, sowie ODE BPEL Compiler, ODE BPEL Engine Runtime und ODE Integration Schnittstelle. Die ODE Runtime-Engine besteht aus den zwei Unterkomponenten ODE DAO (Data Access Objects) und JACOB (Java Concurrent Objects), mit denen die Datenmanipulation und Prozessausführung in ODE realisiert werden können. Die Funktionen in ODE wurden dadurch klar moduliert und mit einander minimal verknüpft. Das realisiert wiederum eine flexible Montage für eine vielfältige Funktionalität des Workflowmanagement-System aus verschiedene Modulen, in denen die einzige und eigenständige Funktion implementiert wurde.

Der ODE BPEL Compiler konvertiert alle eingesetzte Artefakte sowie den BPEL-Prozess, die entsprechende WSDL-Dateien, XML Schemen und deploy.xml. Ein speziales Datenformat für das BPEL Engine Runtime Modul wird generiert. Diese Datei wird durch den Name des BPEL-Prozess Dokument mit der Namenserweiterung ».cbp« identisch benannt. Beim Kompilieren werden die Analyse und Validierung des BPEL-Prozess gegen die standardisierte WS-BPEL Spezifikation durchgeführt. Fehlermeldungen signalisieren falsche oder unvollständige WS-BPEL Syntax und Semantik. In der generierten Datei wird es ein speziales Objektmodell beschrieben, welches eine strukturelle Gemeinsamkeit mit dem eingesetzte BPEL-Prozess besitzt. Durch die Hauptkonstruktor-Klasse »BpelC« und weitere Generator-

Klassen und APIs für verschiedene Strukturen in den BPEL-Prozess und WSDL Dokument wird für das BEPL Engine Modul ein lesbares und ausführbares Modell transformiert. Technische Details des BPEL Compiler und deren Implementierung wird in dieser Arbeit wegen der Komplexität nicht weiter geführt. Motivierte Leser wenden sich an die verwendeten Quellcodes von BPEL Compiler [12].

Das ODE BPEL Engine Runtime Modul ist das Kernmodul der ODE Laufzeitumgebung, die für die Ausführung von den kompilierten BPEL-Prozessen zuständig ist. Die Ausführung von verschiedene BPEL Strukturen bzw. Aktivitäten werden in der BPEL Eingne Prozesslaufzeitumgebung implementiert. In diesem Engine Modul liegen zwei relevante Unterkomponenten JACOB (Java Concurrent Objects) und ODE DAO. Weil die Prozessausführung und Kommunikation sich in einer nicht zuverlässig Netzwerkumgebung finden, soll die Laufzeitumgebung sollen dabei eine Persistenz-Aufforderung für die Instanzen besorgen, um die Prozessausführung in einem sicheren und verlustfreien Verfahren durchführen zu können. Das ODE DAO-Modul beschäftigt sich mit dieser Aufgabe. Die Unterkomponente ODE DAO (Data Access Objects) im BPEL Engine Runtime Modul soll die Interaktion zwischen der BPEL Engine und der extern verbundenen relationalen Datenbank vermitteln und die entsprechende Objektpersistenz bzw. die Datenbehandlungen in der Prozessausführung realisieren. Typischerweise wird dabei eine relationalen Datenbank wie z.B in dieser Arbeit MySQL Datenbankserver [13] genutzt. ODE DAO wird meistens realisiert durch OpenJPA [14], eine eigene frei lizenzierte Implementierung für Java Persistenz API in Apache Software Fundament. Die ODE ODA unterstützt dabei die alltägliche Emissionen von BPEL Engine. Die Zustandsinformationen und ausgetauschten Nachrichten in der Prozessausführung werden in der relationalen Datenbank gespeichert. ODE DAO nutzt die Durchführungsnavigation und Datenzugriffen auf alle Informationen, die durch alle kompilierte eingesetzte BPEL-Prozess und die Kommunikation mit den Außenwelt erzeugt werden. Die BPEL Engine verfolgt und überwacht die Aktivierung von jeder erzeugter Instanz des eingesetzten Prozess, navigiert die kommenden und hinausgehenden Nachrichten für die Prozessinstanzen durch das Korrelation-Set, weist die Input und Output aus den Variablen und Partnerlinks für die Instanz und entsprechende Aktivität hin und synchronisiert insbesondere der Status der ausgeführten Instanz mit dem aktuellem Ausführungszustand in der Jacob virtuellen Laufzeitumgebung.

Die Durchführung von BPEL Basisaktivitäten und strukturellen Aktivitäten für jede Instanz der Prozessausführung werden in der Unterkomponente Jacob (Java Concurrent Objects) Framework implementiert. Jacob bietet einen Thread-unabhängigen Gleichzeitigkeit-Mechanismus für die parallele Prozessausführung und eine effizientes Verfahren für die strukturierte Unterbrechung der Prozessausführung und die Realisierung der Persistenz des aktuell ausgeführten Zustands in der Jacob virtuelle Laufzeitumgebung. Eine technische Erklärung der entsprechende Konzeptionen und Implementierungsmechanismus in Jacob können auf den Weblinks [15] gefunden werden, es wird darin sorgfältig mit BPEL Beispielstruktur anschaulich aufgeklärt.

[12] http://svn.apache.org/repos/asf/ode/trunk/bpel-compiler/src/main/java/org/apache/ode/bpel/compiler/
[13] http://www.mysql.de/downloads/mysql/
[14] http://openjpa.apache.org/
[15] http://ode.apache.org/jacob.html

Das ODE BPEL Engine Runtime Modul implementiert die komplexe geschäftliche Logik, die mehrmals oder ständig das Kommunizieren mit der Außenwelt verlangen. Um eine flexibele Interaktion zwischen dem Engine Runtime Modul und der Außenwelt zu realisieren, es wird eine ODE Integration Schnittstelle montiert und gekoppelt. Im aktuellem Stand werden zwei Integrationsschnittstellem sowie Kommunikationskanäle zur Prozesslaufzeit für Apache Axis2 [16] über das Web Services HTTP Transportprotokoll und serviceorientierte JBI (Java Business Integration) [17] über eine Enterprise-Nachrichtenbus geliefert.

2.2.3. Abstrakte Sicht auf BPEL-Prozess (Business Process View)

Der Geschäftsprozess eines Unternehmens stellt die Kernkompetenz in der Strategieebene in der entsprechenden Branche und die unternehmensweite geschäftliche Tätigkeit in der Organisation dar. Eine gut strukturierter und flexibler Geschäftsprozess lasst die Unternehmen schnell ihre gewinnorientierte Geschäften und Unternehmensstrategien an eine Krise anpassen können. Die Modifizierung und Analyse des bestehende Geschäftsprozess in einem Mittelständischen Unternehmen sind bereits schon eine komplexe Aufgabe, weil die Geschäftsprozesse wegen den vielfältigen Anforderungen viele Aktivitäten beinhalten (wenn es nicht in einer hoch-abstrakte Ebene optimiert wird) und aus mehreren Subprozessen bzw. verschiedenen geschäftlichen Funktionsgebieten integriertet werden. Eine abstrakte Sicht auf Geschäftsprozesse bzw. »Business Process View« sollen dabei helfen, dass die Komplexität eines Prozesses in ein entsprechend betrachteten Dimension reduziert werden können, um eine schnell Analyse und effektive Weiterverarbeitung der Prozessen zu begünstigen.

Eine Abstraktion auf Prozessen steht für eine erwünschte Transformation des bestehender Prozessmodells, dieser Mechanismus wurde durch Angabe von den operative Anweisungen bzw. Regeln implementiert. Eine erweiterbare XML-basierte Regelsprache für die Prozesstransformation wurde in der Diplomarbeit [Str09] erstellt und in der Studienarbeit [Cai10] vervollständigt. In Abbildung 2.3 sind die Definition und die schematisch Aufbauprinzipien der Regelsprache für die Prozesstransformation dargestellt. Die Regelsprache besteht aus der allgemeinen Parametrisierung und den Regeln bzw. Ausweisungen.

[16] http://ws.apache.org/axis2/
[17] http://jcp.org/aboutJava/communityprocess/final/jsr208/index.html

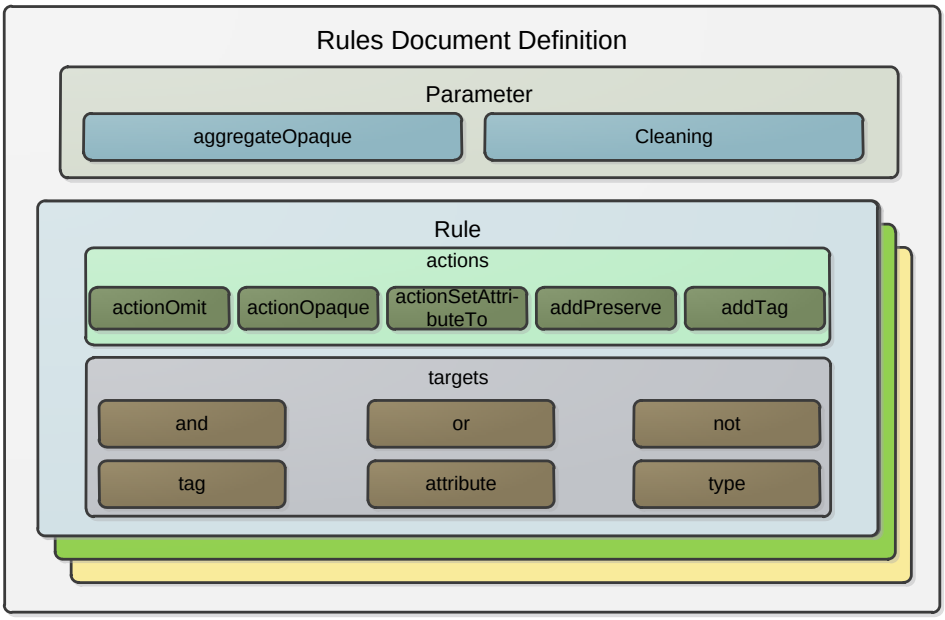

Abbildung 2.3.: Rules Dokument Definition und Komponenten

Ein vollständiges Reglement-Dokument für jede reguläre Prozesstransformation besteht aus zwei Komponenten wie in der obige Abbildung 2.3. Eine vorliegende Anweisung für die Weiterverarbeitung und Bereinigung des Prozesses nach der Transformation wird im Parameterblock spezifiziert. Es werden in der momentanen Applikationsversion zwei Parametrisierungen geliefert. Nach der Angabe der entsprechenden Parametrisierungen werden eine Menge von Regeln bzw. Anweisungen sequenziell in dem XML-basierte Reglement platziert. Es können einzelne oder mehrere Regeln je nach den Anforderung spezifiziert werden. Jede Anweisung besteht aus den erwünschten Aktionen und der Identifizierung des Zielobjektes durch den Aktivitätsname, die Typisierung und weitere Informationen. Für eine ausführliche Erklärung der Parametrisierung und dem Aufbau einer Anweisung können sich an die Arbeit [Cai10, Seite 27, 28, 29] wenden. Es wurde dabei eine detaillierte Beschreibung für die Anwendungsprinzipien und die Zusammensetzung einer Anweisung spezifiziert.

In der Arbeit wurde WS-BEPL Version 2.0 als die grundlegende Spezifikation für die Modellierung von Geschäftsprozess unter Einsatz des Werkzeugs Eclipse BPEL Designer [18] benutzt, die erstellte Prozesstransformation und deren ausführbare Webapplikation. Die aufrufbare Web Services für die Prozesstransformation basieren auch auf den durch WS-BEPL 2.0 implementierten Prozessen. Es ist notwendig, vor der Anwendung der Prozesstransformation diese grundlegenden Voraussetzung für die Basisprozesse zu beachten.

[18] http://www.eclipse.org/bpel/

2.3. Eingesetzte Technologien und Frameworks

2.3.1. Java Servlet Technologie

Ein Java Servlet [19] ist ein in Java geschriebenes Server-seitiges Programm, das die auf dem HTTP-Protokoll basierten Anfragen aus dem Webbrowser verarbeitet und die entsprechenden Antworten zurück senden kann. Durch Servlets können dynamische Inhalte für den Webbrowser generiert werden. Solche Servlets bzw. Java-Klasses werden als nützliche Web-Komponenten zum dynamische Ereugen von Webseiten auf verschiedenen Webservern wegen der Unabhängigkeit von der Laufzeitumgebung erfolgreich genutzt. Ein Servlet und deren Lebensablauf sowie Initialisierung und Löschung von Servlet-Instanzen werden durch einen Servlet-Container bzw. eine Servlet-Engine verwaltet. Die Reaktion zwischen Servlet und Webclient wird durch den Servlet-Container in einen Anfrage-Antworten-Mechanismus implementiert. Der Servlet-Contianer wird in eine, Webserver oder Applikationsserver ständig integriert, um den HTTP-basierten Nachrichtenaustausch über das Internet zu realisieren und weitere Anforderungen, wie Sicherheit und Effizienz zu gewährleisten. Eine ausführliche Erklärung und bisher neueste Spezifikation der Java Servlet Technologie wurden in [Moro9] bekannt gegeben.

2.3.2. JavaServer Pages

JavaServer Pages (JSP) [20] ist eine von Sun Microsystems entwickelte Webprogrammiersprache, mit der gleichzeitig die statische und auch dynamische Webinhalte auf dem Webserver für den Webclient generiert werden. Eine JSP Seite besteht aus zwei Module: Ein Teil erzeugt die statischen Inhalte, sowie Inhalte aus anderen Webprogrammiersprachen für den Webbrowser. Dieser Teil vererbt das gleiche Darstellungsprinzip und die gleiche Grammatik wie HTML oder WML. Im anderen Teil sind alle JSP Sprachelementen, die die dynamischen Inhalte für die Webclient erzeugen können.

Eine JSP Seite behandelt jeweils die Anfragen des Webclients gleich wie ein Servlet. Die JSP Technologie basiert auf der Java Servlet Technologie, deswegen sind die Lebensabläufe und Implementierungsprinzipien einer JSP Seite sehr identisch mit einem Servlet. Wenn eine JSP Seite aufgerufen wird, dann überprüft der Webcontainer bzw. JSP Container das Servlet von dieser JSP Seite. Wenn das Servlet veraltet ist, dann wird der JSP Container transformiert, die JSP Seite nach einen Servlet-Klasse und compiliert den entsprechenden Java-Code. Die Methoden im Servlet werden dann aufgerufen, um die Anfragen vom Webclient automatisch zu verarbeiten und eine dynamisch erzeugte Information für die Webbrowser zu generieren.

Außer den statischen Daten für die Vorlage in den JSP Seiten werden drei Typen von Elementen abgegrenzt. Eine Erklärung von dem jeweiligen Typ wird im Folgenden kurz zusammengefasst:

[19] http://www.oracle.com/technetwork/java/index-jsp-135475.html
[20] http://java.sun.com/products/jsp/

Anweisungen Die Anweisungen in den JSP Seiten beschreiben die globale Informationen für die Transformationsphase. Es liefert den relevante Kontext und die Konfiguration für den JSP-Compiler an. Dazu gehört z.B, das verbundene externe Dokument und die Bekanntgabe von entsprechenden Tag-Bibliotheken. Die Anweisungen verändern den Datenstrom der Antworten überhaupt nicht. In JSP stehen es drei Gruppe von Anweisungen, wie die „<page>" Anweisung, die „<taglib>" Anweisung und die „<include>" Anweisung zur Verfügung.

Aktionen Durch die Aktionen werden die zahlreichen Verarbeitungsmethoden für die Webprogrammierer spezifiziert. Die Aktionen einer JSP Seiten könnten den Datenstrom der Antworten verändern, oder die Objekte des Webservers erzeugen und modifizieren. In JavaServer Pages 2.2 werden viele standardisierte Aktionen in XML-Format angeboten. Es wird auch eigene Aktionen für die spezifizierte Anwendung durch Erweiterungsmechanismen erstellt. Mit Standardaktionen können eine Instanz von Java Beans im Webserver durch „<jsp:useBean>" aufgerufen und durch „<jsp:setProperty>", „<jsp:getProperty>" die Informationen von Beans verarbeiten werden. Mit übertragbaren Tag-Bibliotheken können viele neue Aktionen und Funktionalitäten in den JSP Seiten angewendet werden, wenn die komplizierten Funktionen und Parameter für die Aktionen bereits als „<tag>" Datei realisiert und zur Verfügung gestellt wurden. Solche erweiterten Tag-Bibliotheken können durch die „<taglib>" Anweisung und weitere Attribute sowie Uniform Resource Identifier (URI) in den JSP Seiten verfügbar gemacht werden.

Skript Elemente Skript Elemente in den JSP Seiten manipulieren die Objekten im Webserver und führen die Berechnungen sowie Methoden von Java Objekte durch, um die dynamischen Informationen für die Anfragen zu erzeugen. In JSP 2.0 werden es drei Typen von Skript Elementen sowie „<declarations>", „<scriptlets>" und „<expressions>" erstellt. Durch die Deklaration werden die Variable und Methoden der JSP Seite vereinbart. Außerdem werden alle Deklarationen für die Anwendung von weiteren Skript Elementen initialisiert, erst wenn die JSP Seite erst initialisiert wird. Das Element „<scriptlets>" ist ein kleines Codefragment, welches bei der Verarbeitung von Anfragen ausgeführt wird. Es kann den Datenstrom der Antworten und Java-Objekte im Webserver in entsprechenden Situationen manipulieren. Eine „<expressions>" in einer JSP Seite wird bei der Verarbeitung von Anfragen ausgewertet und das Ergebnis wird als Zeichenkette für den Output in ein JspWriter Objekt konvertiert. Die Zeichenkette sowie das Resultat von „<expressions>" werden dann direkt in die JSP Seiten zur Darstellung eingefügt. Nach JSP 2.1 wird durch die Sprache sowie „<Expression Language>" eine Alternative für die Skript Element angeboten. Die Ausdrücke in den JSP Seiten werden ausgewertet und ebenfalls in eine Zeichenkette als dynamisches Resultat für den Webclient umgewandelt.

Eine ausführliche Erklärung und die bisher neueste Spezifikation von JSP Elementen und weitere Erweiterungen sowie Ausdrucks-Sprachen und Tag-Bibliotheken wurden in [PD09] bekannt gegeben.

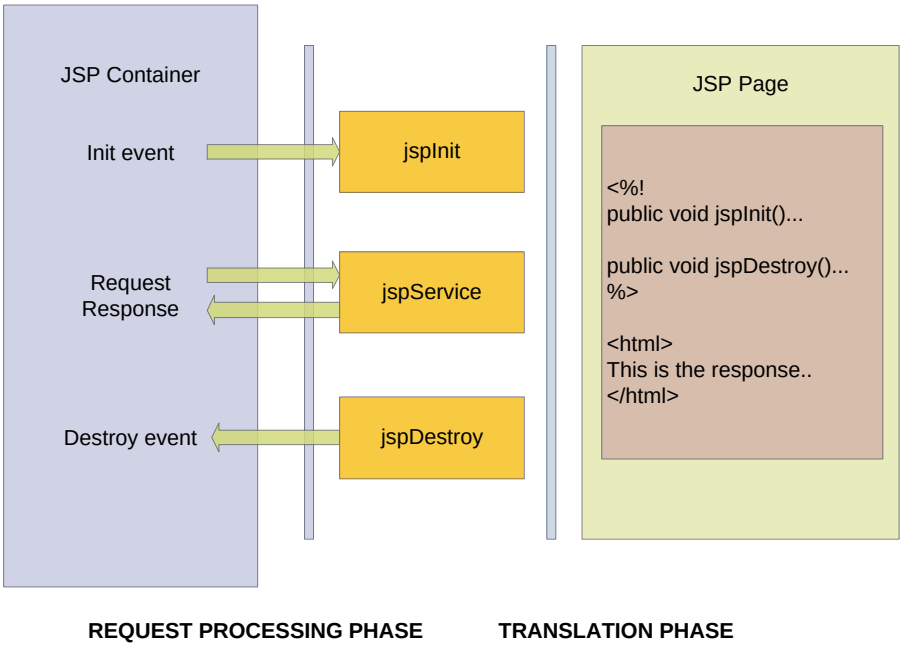

Abbildung 2.4.: Interaktion zwischen JSP Seite und JSP Container

In der Transformationsphase einer JSP Seite werden alle Elemente in einer JSP Seite durch zwei verschiedene Methoden verarbeitet. Die statischen Elemente werden direkt in Code transformiert und in den Datenstrom für die Antworten geschrieben. Die JSP Elemente werden je nach Typ unterschiedlich behandelt. Die Anweisungen werden für die Webcontainer eingelesen, um die Transformation und Ausführung von JSP Seiten zu kontrollieren. Die Skript Elemente werden in der Servlet-Klasse einer JSP Seite transportiert, um den Zugriff und die Manipulation von Java-Objekt zu realisieren. Alle Ausdrücke werden als Parameter für die JSP Ausdruck-Auswerter konvertiert. Die Aktionen einer JSP Seite werden in den entsprechenden Aufruf von Methoden in JavaBeans compiliert.

Nach erfolgreicher Transformation und Compilieren der JSP Seite wird die Servlet-Klasse der JSP Seite durch den JSP Container verwaltet und implementiert. Die Abbildungen 2.4 aus [PD09] stellt das Implementierungsprinzip des JSP Container dar. Die Lebensabläufe der Servlet-Klasse der JSP Seite wird mit den gleichen Prinzip wie in der Java Servlet Technologie behandelt. Wenn eine JSP Seite aufgerufen wird und keine entsprechende Servlet-Klasse existiert, dann ladet der JSP-Container ladet die Servlet-Klasse von JSP Seite nach. Dabei wird eine Instanz der Servlet-Klasse durch den Aufruf der jspInit Methode initialisiert. Der JSP-Container kann mehrfach die jspService Methode aufrufen, um die Java Objekte im Webserver zu manipulieren und entsprechende Datenströme zu generieren. Der JSP-Container ruft die jspDestroy Methode auf, wenn die Servlet-Klasse nicht mehr brauchbar ist.

2.3.3. JavaServer Faces 2.0 Framework

JavaServer Faces (JSF) [21] ist ein auf MVC-Designmuster basiertes Benutzeroberfläche-Framework für die Entwicklung von Java-basierten Webanwendungen. Das JSF Framework konzentriert sich auf den vereinfachten Aufbau von reichhaltigen und handlichen User-Interface für Webapplikation durch die angebotenen UI-Komponenten. Der Entwickler montiert die wiederverwendbare UI-Komponenten in die Webseite und bindet die UI-Komponenten mit der entsprechende Datenressource auf, die aufgerufene oder ausgelösten Events und Funktionen werden auf der Serverseite programmiert.

Eine JSF UI-Komponente ist ein einheitlicher Strukturblock in der Webseite zur Generierung eines JSF interaktives Interface für die Webbenutzern. Dieses interaktive Interface kann eine einfache Eingabe sowie ein paar Knöpfe oder die Textfeldern sein, es kann allerdings auch ein kompliziertes Interface durch zusammengesetzte Strukturen, wie die Baumstruktur oder Tabellen dargestellt sein. Um das interaktive Interface und die unterliegende Datenmodelle mit den assoziierten Java-Objekte zu verbinden und aufzurufen, werden in JSF können es durch die Ausdrücke und Angaben durch entsprechende JavaBeans realisiert. Neben dem interaktiven Interface werden auch zusätzliche Funktionspakete, wie Datentyp-Konvertierung und Eingabe-Validierung entwickelt und geliefert.

Das auf JSF Framewrok basierte interaktive Interface einer Webseite wird als ein »View« für eine besondere Abfrage oder Antwort durch den Zusammenbau der UI-Komponenten erstellt. Eine »View« in der Präsentations-Schicht ist ein hierarchischer Aufbau von Java-Klassen in einer Baumstruktur, die die teilnehmenden UI-Komponenten zur Präsentation implementieren und navigieren. Alle Komponenten in der Baumstruktur werden miteinander assoziiert, jede «View« der UI-Komponenten einer Facette wird dann durch die Wurzelkomponente instanziiert.

Neben der abstrakte Grundklasse für UI-Komponenten sowie »UIComponent« werden auch in JSF solche konkreten UI-Komponenten entwickelt, die den alltägliche Anforderungen des Webclients erfüllen können. Eine detaillierte grafische Auflistung der standardisierten Komponenten können in der Dokumentation [BK09, Kapitel 4] gefunden werden. In JSF generiert es für das interaktive Interface in Webseiten die Events. Nach der Aktivierung von Ereignisses werden die Events behandelt und die entsprechenden Antworten zur UI-Komponente in »View« zur Darstellung abgeschickt.

Das Implementierungsmodel für Events in JSF ist sehr ähnlich wie die Implementierungsmethode in gängigen Benutzeroberflächen-Anwendungspakete z.B das Swing Framework in JDK. Der Empfänger der Events wirdn registriert und durch entsprechende Events aktiviert. Der Empfänger ruft die spezifizierten Java-Klassen auf, um die unterliegende geschäftliche Logik zu implementieren. In den UI-Komponenten und deren Subklassen wird das Event herausgegeben, welches eine maßgebliche Zustandsveränderung in der Anwendung besagt. Diese Nachricht wird dann im zentralen Register verbreitet, in dem viele Empfängern sich angemeldet haben, welche einen bestimmten Eventtyp zu verarbeiten wollen. Es kann zu dem Fall kommen,dass die Events in einer Schlange für einen Empfänger bereitstehen.

[21] https://javaserverfaces.dev.java.net/

2.3.4. Hibernate Framework

Hibernate [22] ist ein leistungsfähiges und flexibeles Programmiermodell für die Persistenz und Anfragen von Datenressourcen, mit der die Java-Objekte und deren Attribute in eine Applikation mit den relationale Datenschemen von Datenbank zusammen abgebildet werden, um die Entwicklungskosten und aufwändige Programmierung zu vermindern. Die Kernaufgabe von Hibernate ist die effiziente Abbildung zwischen Objektorientierter Programmierung und relationaler Datenbank, es verknüpft die Java-Klassen sowie Plain Old Java Objects (POJO) mit den Datentabellen. Die jeweiligen Datentypen von Java Objekten werden mit entsprechende Datentypen der SQL Datenbank verbunden.

Neben der Objekt/Relation Abbildung bietet Hibernate weitere effiziente Funktionalitäten für die Datenabfrage an, die den Entwicklungsaufwand von Anwendung und Datenbank deutlichen verbessert. Die Datenabfragen durch jeweils überflüssige Codierung von SQL und Java Database Connectivity (JDBC) sind wegen häufigen Aufrufen von Datenzugriffen aufwändiger. Hibernate realisiert ein verbessertes Implementierungsprinzip für solche Tätigkeiten, die ständige mit Persistenz von Datenressourcen zu tun haben. Der große Teil der Codierung durch SQL-Ausdrücke und JDBC für die Datenverarbeitung wird durch entsprechende Mapping-Metadaten von Hibernate realisiert und die relevanten Interaktionen zwischen Anwendung und Datenbank werden transaktional verwaltet.

Persistenz Hibernate wird als eine kompetente Lösung für Persistenz von Objekt und Datenressourcen in Applikation betrachtet und weiterentwickelt. Unter Persistenz in dieser Arbeit versteht man die theoretische langfristige Einbehaltung eines Datenzustand des Objektes nach der Erzeugen durch ein Programm oder Java-Klasse. Der aktuelle Zustand eines Objekts wird verfügbar für weitere Prozesse, die diesen Informationszustand des erzeugten Objekts in einem bestimmtem Zeitraum weiter benutzen wollen. Solche Objekte und Datenressourcen in Applikation bzw. der gespeicherte Informationszustand beispielsweise im Cache können durch Aufrufen von spezielle Methoden gelöscht werden. In dieser Arbeit wird die physikalische Persistenz aus Datenbankmanagementsystem nicht diskutiert.

Hibernate vermindert die wiederholte Codierungen zwischen Applikationsfunktionen und Datenbankzugriffen. Die Softwareentwickler konzentrieren sich auf die relevante Aufgaben der Geschäftslogik von Applikationen und benutzen bekannte Programmiermuster, um die Datenverarbeitung und Datenabfrage effizienter zu realisieren. Die Anwendungsmöglichkeit von Hibernate hängt nicht von den Softwareentwicklungsparadigma bzw. den Entwurfsstrategien ab. Es zeigt sich eine große Einsatzfähigkeiten für populäre Anwendungssysteme in alle Branchen, die besonders mit großen und deutlich häufiger verarbeiteten Datenmengen umgehen und dabei eine höhere Datenintegrität gewährleisten müssen. Eine Weiterentwicklung von bereits bestehenden Datenschemen in der Datenbank oder eine Entwicklung von einem neuem Geschäftsanwendungssystem wird dabei deutlich beschleunigt durch die Einsatz von Hibernate sowie die Objekt/Relation abgebildete Persistenz-Lösung zwischen Anwendungsschicht und Datenschicht.

[22] http://www.hibernate.org/

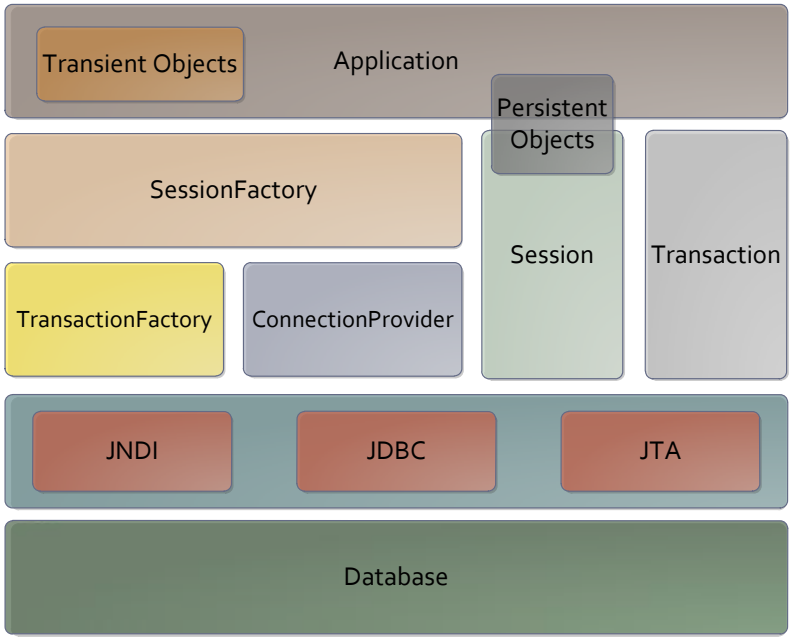

Abbildung 2.5.: Detaillierte Hibernate-Architektur aus [GKE10]

Hibernate schweißt die zwei verschiedene Repräsentationen von Daten in Anwendungsentwicklung und Datenbankmodellierung zusammen. Die Kompetenzen von beiden Seiten werden nicht von dieser hinzugefügten Klebeschicht beschädigt und verlangsamt, sondern werden durch bessere Mechanismen optimiert . Die Abbildung 2.5 stellt diese Klebeschicht zwischen der Applikation und der Datenbank graphisch dar. Die wichtigen Komponenten aus Hibernate und ein Überblick werden danach kurz gegeben. Eine ausführliche Funktionsbeschreibung von relevanten Bauelementen in Hibernate können in [GKE10] gefunden werden.

SessionFactory Eine Session Fabrik wird in Hibernate durch die Konfigurationsdatei bei der Instanziierung erzeugt. Es wird für eine einzelne Anwendung und eine einzelne Datenbank nur eine Session Fabrik spezifiziert. Eine Session Fabrik besitzt Threadsicherheit. Die Ausführung wird durch den gleichzeitigen Aufrufen von anderen Programmen nicht behindert. Alle Sessions und die Erfassungsarbeit von Metadaten werden durch die Session Fabrik zur Laufzeit erzeugt und verwaltet. Die Eigenschaften von Session Fabrik sind nach den Instanziierung in Konfigurationszeit unveränderbar. Durch das Schließen einer Session Fabrik werden alle Ressourcen in einen sekundären Cache und die Anbindungen der Datenpools befreit.

Session Eine Session ist eine kurzfristige Konversation zwischen der Applikation und der Datenpersistenz in Hibernate. Die Anbindung mit Datenbank durch JDBC und Datenabfragen durch SQL-Ausdrücke wird in jeder Session von Hibernate verpackt. Die transaktionale Anforderung der Datenverarbeitung wird in der Session gesichert

und kontrolliert. Der gesamte Persistenz-Kontext von Java-Objekt wird in der Session zwangsweise in einem primitive Cache gespeichert. In einer Hibernate-Session werden die relevanten Operationen für die Modifizierung und Persistenz von Java-Objekten angeboten. Das Durchsuchen der richtigen Objektidentität und der entsprechend assoziierter Instanz der Session im Cache sollen dabei die korrekte Persistenz des Informationszustand des jeweiligen Objektes navigieren und sichern.

Persistent objects Das Java-Objekt und die Kollektion von Objekten, die den Persistenz-Kontext bzw. die »Hibernate Annotations« und die geschäftliche Anwendungsfunktionen beinhalten, werden eindeutig assoziiert durch genau eine Session-Instanz. Nach der Datenverarbeitung und den Funktionsaufrufen des Persistenz-Objekts wird die assoziierte Session abgeschlossen, um den aktuellen Informationszustand des Java-Objekts zu speichern und für die weitere Verarbeitung von anderen Programmen und Prozessen in der Anwendung freizuschalten.

Transient and detached objects Die Persistenz-Objekte werden nach dem Abschließen der assoziierten Session losgelöst. Solche frühere Persistenz-Objekte werden dann »detached objects« geworden, die mit keiner Session und Instanz mehr verbunden sind und keinen Persistenz-Kontext mehr besitzen. Die Java-Objekte werden durch Java-Klasse initialisiert und als kurzlebige Objekte wie »Transient objects« im Cache gespeichert. Das gilt für Objekte, die nicht als Persistenz-Objekte durch Hibernate behandelt werden.

Für die Objekt/Relation Abbildung braucht Hibernate die Metadaten, die die Transformation von Informationsrepräsentation aus Java-Objekt in eine relationale Repräsentation in Datenbankschicht navigieren und leiten. »Hibernate Annotations« sind in das Java-Objekt bzw. direkt vor die Codezeile der Java-Klasse durch das »@« Zeichen und weitere Attribute unmittelbar kommentierte Metadaten. In »Hibernate Annotations« können zwei verschiedene Kommentierungs-Prinzipien klassifiziert werden. Entweder durch die objektorientierte Modellierung, in der die Eigenschaften von Objektklasse und die Beziehung zwischen mehrere Objektklassen kommentiert werden, oder nach den Datenschemen in den Tabellen, welche die entsprechende Spalte und Index bzw. den Schlüssel im Java-Objekte durch einen Kommentar abgebildet werden. Es können in der Praxis die beide Prinzipien gemischt angewendet werden.

In EJB 3.0 bietet neben den Annotationen noch ein neue Methode, den »XML deployment descriptor« als eine weitere Möglichkeit an, mit der die durch »Java Persistenz Annotation (JPA)« vorbereiteten kommentierten Metadaten in der Java-Klasse überschreiben oder ausgetauscht werden können. Die Eigenschaften und die Assoziationen der Java-Klassen, wie »Entity« im Persistenz-Kontext können durch den »XML deployment descriptor« einfach bei der Stationierung von Metadaten bedeckt werden. Im Dokument [Ber10] können eine detaillierte Spezifikation zusammen mit ausführlichen Codebeispiele von verschiedene Kommentierungen bzw. Metadaten und die Anwendung der entsprechenden Attributen in JPA für die Persistenz-Klassen gefunden werden.

2.3.5. Spring Framework

Das Spring Framework [23] ist eine Java-basierte Plattform für die Entwicklung von unternehmensweiten leichtgewichtigen Anwendungen zusammen mit den gängigen JEE-Technologien. Es bietet kompakte Grundbausteine und eine umfassende Infrastruktur zur Unterstützung aller möglichen Java-Anwendungen an. Das Spring Framework beschäftigt sich mit der Infrastruktur und der Wiederverwendbarkeit von den Applikationskomponenten in der Enterprise-Anwendung. Durch die Anwendung von Spring können die Softwareentwickler bei der Entwicklungsphase unabhängig erst mit der Programmierung von POJOs bzw. der Datenmodellierung für der Datenbank anfangen und dann die Geschäftslogik sowie Services und deren Implementierungen als Java-Beans mit entsprechende Metadaten in den Anwendungscontainer konfigurieren, ohne die Java-Beans und Klassen angreifende zu durchdrängen bzw. assoziieren.

Das Spring Framework beinhaltet vielen Merkmale und Funktionen, die in mehrere Module und Programmpakete klassifiziert werden. Die relevanten Module in Spring sind die Kerncontainer, »Data Access/Integration«, das Webmodul, »Aspect Oriented Programming", Instrumentation und Testmodule. Diese sind graphisch in der Abbildung 2.6 aus Spring Dokumentation [RJ+10, Seite 3] schematisch dargestellt. Die nachfolgende Erklärungen von Modulen und Funktionen des Spring Frameworks werden im Großteil theoretische aus Dokumentation [RJ+10] für einen unkomplizierter Einführung in die Themen persönlich zusammengefasst. Die Leser können sich an die Dokumentation und weitere online Artikeln [24] wenden, um ein technisch vertieftes Verstehen des Spring Frameworks zu erzielen. Außer den Modulen und der Architektur sind noch zwei weitere relevante Konzeptionen aus Spring Framework zur Vorstellung notwendig. Sie werden dann in den folgenden Abschnitten für einen technischen Überblick kurz zusammengefasst.

Inversion of Control Das ist eine umgekehrte Methode für das Aufrufen von Komponenten in dem Programm, ohne die direkt Initialisierung, Erzeugung, Verdrahtung und Verwaltung des Objektes durch das Hauptprogramm. Die Konfigurationsaufgabe von allen verbundenen Objekten in den Komponenten werden durch die außen Container in Spring Framework geliefert und verwaltet. Dabei nimmt das Spring Framework die Rolle des Hauptprogramms ein und kümmert sich um den Lebenszyklus des Objektes. Es wird dabei geholfen, um die Kopplung und strenge Abhängigkeit zwischen Komponenten zu verringern. Die alle Nachschlagen von zielte Komponenten und Konfigurieren des Applikationskontextes werden ins Container gezogen. Durch das Verfahren »Dependency injection« sind Container verantwortlich für die Konfiguration und Verdrahtung von Komponenten. Eine konkrete Implementierung der Konfiguration und Verdrahtung werden in die Klassen eingespritzt. Ohne eine große Codeveränderung der Klassen werden der verbundene Kontext und die Verdrahtung in den Container verändert und neu konfiguriert.

Aspektorientierte Programmierung Das ist ein weiteres Programmierparadigma mit Unterschieden gegenüber der Objektorientierten Programmierung, durch den die funktionale

[23] http://www.springsource.org/
[24] http://www.springsource.org/documentation

Aspekten der Softwareanwendung methodische separiert werden und um sie getrennte zu entwickeln und zu testen. Ein Aspekt hier ist die eigenständige Funktionsanforderung in der Anwendung, die nicht als ein komplettes und relativ selbständiges Modul in der OOP entwickelt wird. Ein Aspekt wird sich in der ganzen Anwendung einen querschnittlich Belang bezeichnet, weil er nicht modularisiert wird. Eine Logging-Funktion z.B. greift auf alle Module einer Anwendung zu. In der aspektorientierten Programmierung werden solche Aspekte als syntaktische Strukturen in der Anwendungsentwicklung im Bezug modularisiert und dann mit den restlichen Funktionsmodulen verwoben.

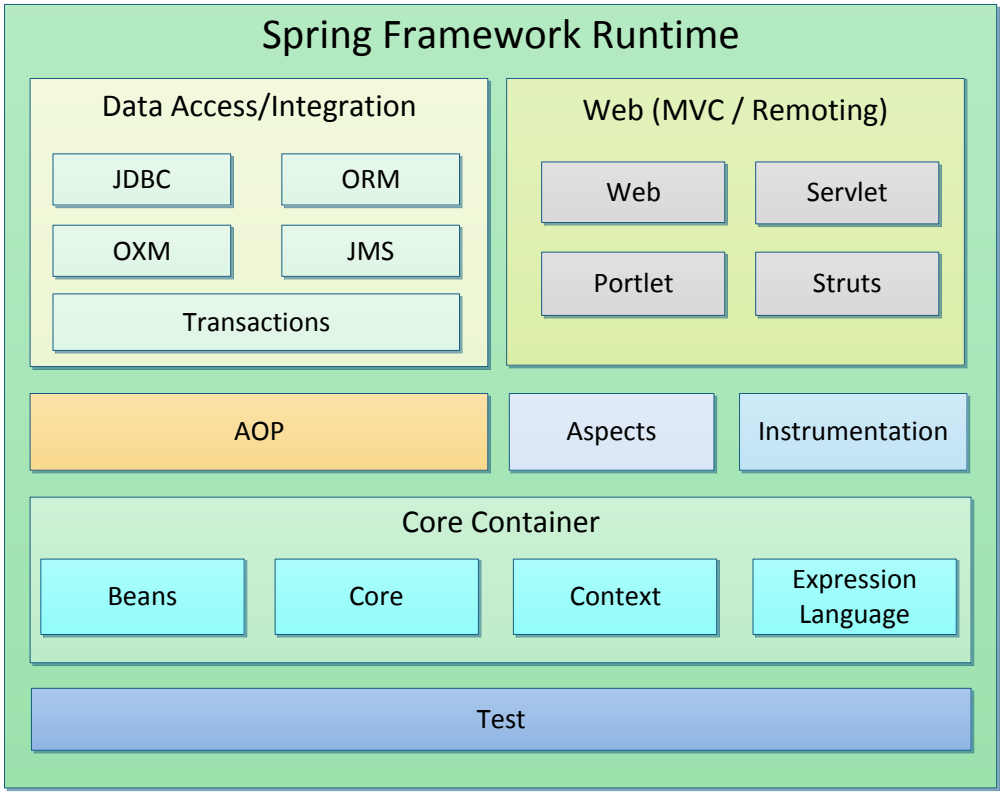

Abbildung 2.6.: Ein Überblick von Spring Framework aus [RJ+10]

Der Spring Kerncontainer beinhaltet die vier fundamentalen Module, wie in der obige Abbildung angezeigt. Das Kern- und Beansmodul sind die grundsätzliche Funktionspakete des Framworks. Die Hauptfunktionen sowie die »Inversion of Control« und deren Implementierungsmethode »Dependency Injection« werden darin angeboten und verwaltet. Das org.springframework.beans.factory Paket unterstützt die Verwaltung und Manipulation von Beans. Das Kontextmodul baut die Funktionalitäten von Kern- und Beansmodul in einem framework-orientierte Stil weiter aus, es bietet die Anpassung für die Internationalisierung

und das Ressourcen-Aufladen neben der ererbten Funktionen aus dem Beansmodul an. Das Ausdrücksprache-Modul realisiert die Anfrage und das Manipulieren von Java Objekten zur Laufzeit und ist ein erweitertes Paket von »unified expression language« aus der JSP-Spezifikation. Es können das Datenobjekt und die Methodenklasse und deren Merkmale durch die Methoden »Setter« und »Getter« ausgewertet, abgeholt und aufgerufen werden. Die Objekte werden bereits im Kerncontainer registriert, das erwünschte Objekt wird dann durch die Ausdrücke wieder erstellt.

In der Datenzugriffs und -Integrationsschicht befinden sich die Module für JDBC, OR-Mapping, OXM, Java Messaging Service (JMS) und Transaktionsmanagementmodell des Spring Frameworks. Es werden auch die Funktionspakete für die MVC-Designmuster basierten Webanwendungsentwicklung und Entwicklungsplattform-Integration sowie das Webmodul und die Module für Web-Servlet, Web-Struts, Web-Portlet in Spring entwickelt. Test ist eine wichtige Arbeitsphase in der geschäftlichen Anwendungsentwicklung, das unterliegende Testmodul im Spring Framework unterstützt dabei zwei Verfahren mit entsprechenden Testpaketen und Objekten zur Testkonfiguration. Die Einheitstest und die Integrationstest durch die Tools z.B, JUnit und TestNG werden die Qualität von Anwendung in verschiedene Dimensionen verbessern und sichern. Für die ausführlichere Spezifikation und Erklärung von den weiteren Merkmalen und Funktionen im Spring Framework können umfangreich in [RJ+10] gefunden werden. Es wird in dieser Arbeit nicht alle Merkmale und Technologie im Spring Framwork außer die relevante Konzeptionen weiter vorgestellt und diskutiert.

2.3.6. Apache Struts2 Framework

Apache Struts2 [25] ist ein elegantes und erweiterbares Framework für die Entwicklung, Stationierung und Wartung von auf Java-basierenden geschäftlichen Webanwendungen. Durch den Einsatz von Web Frameworks und Entwicklungsplattformen werden die meist identischen Aufgaben in der Webanwendungsentwicklung automatisiert und den Entwicklern wird dabei geholfen, sich auf die wichtigen Aufgabe bzw. die geschäftliche Logik der Geschäftsanwendung und anderer relevanter Funktionsanforderungen, wie Datensicherheit und Robustheit zu konzentrierten. Apache Struts2 befolgt und implementiert bekannte Designmuster für die Webanwendungsentwicklung, wie z.B. Model-View-Controller. Die Präsentations-Schicht in der Webanwendungsarchitektur werden durch JSP-Technologie und konfigurierbare UI-Tags von Struts2 realisiert. Die Antworten auf die entsprechende Abfrage werden durch die Aktion sowie den »Controller« generiert und zum Webclient versendet. Die Abfragen und inhaltliche Parameter des Webclients werden in Struts2 gefiltert und zum nächsten »Dispatcher« weitergeleitet. Struts2 kontrolliert diesen Vorgang über mehrere Verteiler bis zur richtigen Aktionsklasse. Die Parameter und Abfragen werden schrittweise durch den »Interceptor« abgefangen. Er kann die Eingaben validieren und nach den erfolgreiche Aufrufen von Aktionsklasse weiter verarbeiten bzw. protokollieren. In den Aktion wird die entsprechende geschäftliche Kommunikation und Aufrufe von Prozeduren

[25] http://struts.apache.org/2.x/index.html

zwischen der Datenbank und POJO Klassen ausgeführt und die Antworten als Ergebnis zum Webclient zurückgeschickt.

Struts2 bietet viele allgemeine XML-Elementtypen, wie »Tags« an, die vielfältige Funktionen in den JSP-Page schaffen kann. Die generischen Tags kontrollieren beim den Rendering von JSP-Page den Ausführungsprozessen. Es werden dabei die Daten und Parameter in den Aktionsklassen oder JavaBeans ausgewertet, modifiziert und ausgeholt für das HTML-Rendering. Mit dem Tag »Form« können in Struts2 die verschiedenen Daten einfacher eingegeben werden. Die entsprechende Datenverarbeitung durch die Aktionsklasse wird in den Attributen des Tag-Element durch die Angabe von Aktionsname spezifiziert und annotiert. Mit dem Tag »Property« können die Variablen durch Aufruf von einer »Getter« Methode in Aktionsklasse ihr Inhalt ausgewertet und für die Webpräsentation abgeholt werden. Die UI-Tags in Struts2 sind unterscheiden sich von den generische Tags. UI-Tags werden dabei genutzt, um die Daten in einer reichhaltigen HTML-Präsentation anzuzeigen, die von verschiedenen Themen und Darstellungsstilen abhängig sind.

Eine Aktion in Struts2 zu erzeugen schließt drei Schritten ein: Als erstes die Abbildung zwischen Aktion und Klassen, dann die Navigation für die Resultate und letztlich die Implementierung von der Geschäftslogik. Eine Aktion handelt in der Rolle »Controller« im obengenannte MVC-Designmuster. Die geschäftliche Logik wird in der Aktionsklasse oder in anderen durch diese Aktion aufgerufene Klassen durchgeführt. Jede Aktion in Struts2 besitzt einen eindeutigen Namen, welcher durch Angabe vom Klassenpfad auf die entsprechende Klasse abgebildet wird. Die Resultate der Aktionen werden in einer bestimmte JSP-Page sowie im »View« präsentiert. Dabei wird auf die nachfolgende Darstellungswebseite durch die Aktion navigiert. Für jede Aktion wird eine Dokumentation in der Konfigurationsdatei »struts.xml« angegeben. Die Implementierungslogik muss weiter in der Aktionsklasse realisiert werden. Darin wird mindestens eine ausführbare Methode mit spezifizierter Funktion und vordefinierter Zeichenkette-Ausgabe definiert. Die generische Aufgabe der Aktionen ist die Verarbeitung von Eingabedaten. Dazu arbeiten die Aktionsklassen und »Tags« in Struts2 zusammen, um die Daten zwischen Präsentations-Schicht und Geschäftslogik zu transportieren. Durch Angabe und Assoziation von Aktionsname und Merkmal in den »Tags« Elementen werden die Aktionsklasse nach der Abgabe automatisch die Daten und Werte für die Eigenschaft in JavaBeans durch die »Setter« Methode weiterleiten. Eine Datenspeicherung und -Persistenz können auch innerhalb einer Aktionsklasse implementiert werden.

2.3.7. Apache Axis2 Framework

Apache Axis2 [26] ist ein auf dem ursprünglichen Apache Axis basierte und weiterentwickelte Web Services Engine der dritten Generation. Es macht die Implementierung von Web Services effizienter und das eigene Framework sowie die Infrastruktur modularisierter. Außerdem sorgt es für die Konzentration und Orientierung auf XML-Konzepte in der Programmierung und Realisierung von Web Services strebt die Apache Axis2 auch an. Die entworfene Architektur von Axis2 unterstützt dabei die höhere Erweiterbarkeit und

[26] http://ws.apache.org/axis2/

Kompatibilität für die Addition von neue Modulen, die einen erneute Funktion oder Merkmal für Sicherheit und Zuverlässigkeit ausweiten können, um die vielfältigen und gestiegenen Kommunikationsanforderungen zwischen Web Services und Anwendungsbenutzern zu erfüllen.

Die Kernaufgabe von Apache Axis2 ist die Java-basierte Verarbeitung und Transformation von XML-basierten SOAP-Nachrichten zwischen dem Service-Aufrufer und dem Service-Anbieter. Die Verarbeitungsgeschwindigkeit und die Effizienz des Speicherverbrauchen des Speichers sind sehr hoch durch die angewendete neue XML-Verarbeitungsprinzipien im Vergleich zu den vorangegangenen Web Services Engins, wie Apache Axis oder Apache SOAP Projekte. Mit Axis2 können komplexe SOAP Nachrichten gesendet, erhalten und verarbeitet werden. Die Implementierungsklassen von Web Services können durch die vordefinierte Spezifikation in der WSDL Datei automatisch geführt und dann die entsprechenden Web Services in Server einfacher produzieren werden. Neben der lose gekoppelten Modularisierung werden weitere Tools zur Unterstützung der Entwicklung des Web Services z.B, WSDL2Java und Java2WSDL entwickelt. Dadurch werden der Entwurf und die Implementierung von Web Services erleichtert. Um einen hohe Interoperationalität in der Werservices-Welt zu erreichen und zu unterstützen, werden viele technische Spezifikationen und Empfehlungen z.B WS-Security, WS-ReliableMessaging, WS-Addressing, WS-Coordination vorgestellt und weiterentwickelt. Diese fordern große Funktionalität und flexible Kombinationsmöglichkeiten für die Verarbeitung von SOAP Nachrichten. Die strukturelle Modularisierung von Apache Axis2 kann sich schnell an diese Anforderung anpassen und neue Module reibungslos in den Nachrichten-Verarbeitungsprozess sowie den verbundenen SOAP-Kanal integrieren.

Die in Axis2 angewandte XML-Objektmodell sowie AXIOM (AXIs Object Model) [27] verbessert die Verarbeitungsproduktivität innerhalb der inhaltlich Recherchen von XML-basierte Nachrichten sowie die Repräsentation des Infosets im XML Dokument. Es benutzt den »Ziehen« Mechanismus in der Erzeugung von XML-Objekten und basiert in der technischen Implementierung auf die StAX (Streaming API for XML) [28] Parser. Es passiert immer in den Alltag, dass die SOAP-Nachrichten extrem groß sind und informationell dicht eingebunden in unsere Geschäftsanwendung sind. Eine komplette Repräsentation des XML-Dokuments im Arbeitsspeicher oder Cache ist nicht mehr sinnvoll und verursacht ein Kapazitätsproblem und führt zur Verlangsamung. Der StAx Parser kontrolliert diese Repräsentation und verzögert das Darstellungsereignis des XML-Dokument. Es wird ein unnötiges Einlesen von ungebrauchten Informationen vermieden. Z.B bei weiterem Transportieren von SOAP-Nachrichten wird in diesem Fall nur die Kopfteile der SOAP-Nachrichten ohne den Hauptteil der SOAP Nachrichten auf die Informationen untersucht. Dadurch wird Kapazität auf dem Webserver gespart. Eine technische Analyse der Kapazität und Geschwindigkeit von XML-Parser können in [Tea05] gefunden werden. Es zeigt einen deutlichen Vorteil beim Einlesen und bei der Untersuchung von XML Dokumenten ohne die Anforderung für die Modifikation des Dokuments gegenüber anderen XML Parser.

Ein müheloser Einstieg in Axis2 und viele neue Funktionen und entsprechende Beschreibungen von Apache Axis2 sowie die »SOAP Processing Model and Pipeline«, »Message

[27] http://ws.apache.org/commons/axiom/
[28] http://jcp.org/en/jsr/detail?id=173

Exchange Patterns« und »Axis2 Data Binding« können in der Projektweblink [29] online gefunden werden.

[29]http://ws.apache.org/axis2/articles.html

3. Konzept und Entwurf

Im vorliegenden Kapitel wird auf das erstellte Konzept und den Entwurf für die Realisierung der Verwaltungsplattform eingegangen. Die grundlegende Konzeption und die abstrahierte Architektur für den Entwurf und die Implementierung werden im ersten Abschnitt 3.1 vorgestellt. In der Praxisarbeit wird eine Webanwendung für die Prozess View Transformation auf den Webservices-Konzepten basierend weiterentwickelt. Der spezifizierte Entwurf und die zugehörigen Module werden im zweiten Abschnitt 3.2 erläutert. Im Abschnitt 3.3 werden das innerhalb der Arbeit erstellte Neukonzept für den Prozess View vorgestellt, mit dem eine benutzerdefinierte Prozess View Vorlage kombiniert und ausgeführt werden kann.

3.1. Konzept und Architektur

Ein durch Java EE Version 5 als Programmiersprache implementiertes Jar-Paket für die konsolenbasierte Durchführung der Prozess View Transformation wurde bereits entwickelt. Eine grundlegende Idee ist die Entwicklung einer Webanwendung, die dieses Jar-Paket hauptsächlich für die Realisierung der Prozesstransformation nutzt und die weitere Anforderungen z.B hohe Modularität und Bedienbarkeit erfüllen kann.

Die Webanwendung realisiert ein benutzerfreundliches User-Interface für die Prozesstransformation. Es bietet ein einfaches Abgeben des Prozessdokuments bzw. des BPEL-Prozess und des entsprechenden Rules-Dokument. Ein verarbeitender Prozess wird nach der Durchführung der Prozesstransformation auf der Serverseite z.B in Tomcat generiert und im den Browser präsentiert. Es wird ebenfalls unterstützt, dass nach dem Prozesshochladen die verfügbare View-Funktionen in [Cai10] flexible benutzt werden kann.

Neben dem BPEL-Prozess ist das Rules-Dokument ein kritisches Artefakt. Konventionell wird das Rules-Dokument manuell handwerklich erstellt. Es wird ebenso überlegt diese Produktivität zu erhöhen. Ein webbasierter Editor für das Rules-Dokument soll entwickelt werden, dessen realisierbare Funktionalität sich an der kompletten Spezifikation in der Arbeit [Stro9] richten soll.

Die jeweils durchgeführte Transformation wird auf der Serverseite für den Administrator protokolliert. Die Anwendung registriert neben den abgegebenen Daten, den Prozessen und Rules auch die Datumsinformation. Es erschafft dem Administrator eine Datenbasis für die Performance-Analyse und Fehlerprotokollierung. Es kann wegen der schnellen Steigerung der Kapazitätsanforderung eine Datenabschaffung durch den Administrator durchgeführt werden.

Es wird in der Arbeit die Konzeption von Prozess View erweitertet. Es entwickelt ein neues Anwendungsszenario für Prozesstransformation. Die bestehende View-Funktionen in [Cai10] werden als aufrufbarer Webservices für Prozess View implementiert und veröffentlicht. Die grundlegende Operation und Zielnavigation werden ebenfalls als Webservices realisierbar. Die Webanwendung bietet die verschiedenen Webservices für die Prozesstransformation an. Alle verfügbaren Webservices werden zentral verwaltet und stellen eine Servicebasis für die Konstruktion von komplexen Sichten auf BPEL-Prozesse bereit.

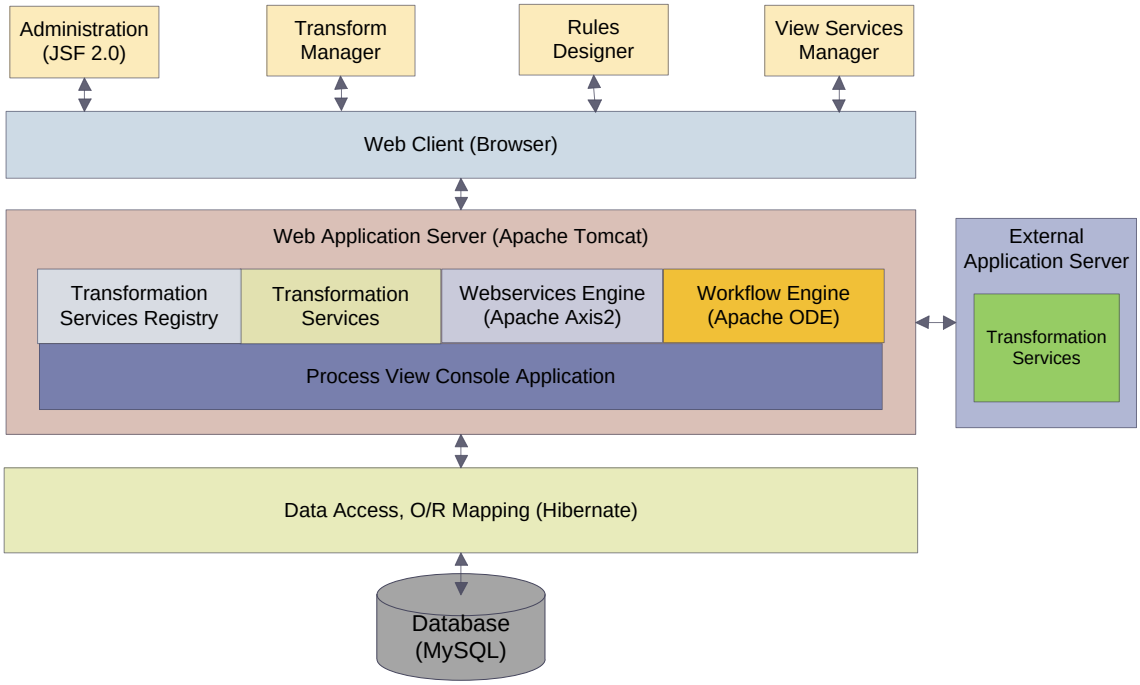

Abbildung 3.1.: Business Prozess View Verwaltung Architektur

Die Abbildung 3.1 zeigt die Mehrschichtarchitektur der Webanwendung für die Implementierung. In der untersten Schicht liegt eine relationale Datenbank wie z.B MySQL. Dort wird jede Prozesstransformation der Serverseite der Webanwendung gespeichert und protokolliert. Das angewandte Persistenz-Framework wie z.B Hibernate für die Datenzugriffe bietet eine bessere Performance als andere Datenbankschnittstellen-API z.B JDBC. Es erzeugt das Datenobjekt und manipuliert es ohne eine explizite Programmierung von SQL-Ausdrücken. In der Applikationsschicht liegt die Webanwendung. Sie beinhaltet das Kernmodul für die Prozesstransformation und für die als Webservices implementierte, zentral registrierte Prozess View Services. Sie arbeitet zusammen mit der Webservices Engine für die Servicekommunikation und mit der Workflow Engine für die Erzeugung des WS-BPEL basierten Webservices. Die Präsentationsschicht besteht aus vier unterteilten Web-Interfaces für die Benutzer und den Administrator.

3.2. Entwurf

In diesem Abschnitt werden die entworfenen Anwendungsmodelle und -Diagramme der Webanwendung beschreiben. Der View Services Manager und die erstellte View Services Architektur werden wegen dem Kerngewicht in der Arbeit detaillierter erklärt.

3.2.1. Anwendungsmodellierung

Es wird zuerst ein Anwendungsmodell für eine webbasierte Prozess View Verwaltungsplattform in der Entwurfsphase erstellt. Sie dient als eine Basis für die Entwicklung des User-Interfaces und der gestützten logischen Funktionen. Im Anwendungsfalldiagramm 3.2 sind die erforderlichen Hauptanwendungsfälle und die beteiligten Akteuren strukturiert dargestellt.

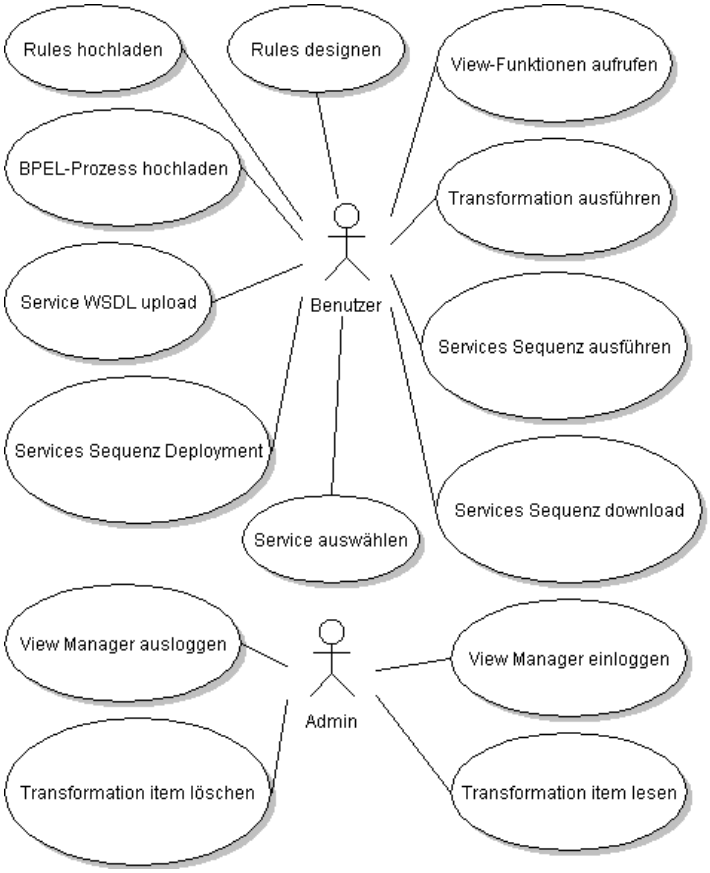

Abbildung 3.2.: Anwendungsfalldiagramm von der webbasierten Verwaltungsplattform

Die einzelnen Begriffe des Diagramms werden kurzgefasst erklärt, um dem Leser einen einfaches Verständnis des Diagramm zu schaffen und eine klare und nicht überlappende Klassifizierung der Anwendungsfälle zu gliedern.

View-Funktionen Die View-Funktionen sind die zwei bereits feststehenden Funktionen Fokussierung und Subprozesseliminieren für eine vordefinierte Prozesstransformation. Durch das Hochladen der entsprechenden Parametern werden dann diese Funktionen aufgerufen und die Prozesstransformationen werden durchgeführt.

Transformation Transformation sind die Prozesstransformation, die durch das Hochladen des BPEL-Prozess und des Rules Dokuments generiert wird. Dieser Vorgang wird von dem Benutzer ausgelöst.

Services Sequenz Services Sequenz ist hier eine aus einer einzigen Operation oder aus mehreren Operationen sequenzielle, kombinierte Warteschlange von der später ein kompletter BPEL-Prozess erstellt wird.

View Manager View Manager ist hier das zentrale Webinterface für die Datenbanktabelle, in der alle existierenden Datensätzen in der Registrierung angezeigt werden. Es ist aber nur für den Administrator zugänglich.

Transformation Item Transformation Item ist hier der gespeicherte Datensatz in der Registrierung. Er resultiert von den durchgeführten Anwendungsinstanzen der Prozesstransformation. In jedem Transformation Item werden die relevanten Artefakte wie der BPEL-Prozess, das abgegebenen Rules Dokument, das View-Ergebnis und die assoziierte Zusatzinformationen registriert.

Alle entworfenen Anwendungsfälle werden im Folgenden jeweils einzeln beschreiben:

Anwendungsfall 1: BPEL-Prozess hochladen

Aktoren:

Der Benutzer

Vorbedingung:

Die Seite „View Transformer" im Webinterface wird durch den Klick angezeigt und darin wird ein Subinterface „BPEL-Process Upload" angezeigt.

Regulärer Ablauf:

1. Der Benutzer kopiert den kompletten Quellcode des BPEL-Prozesses ins Textfenster.
2. Der Benutzer klickt den Button „submit".

Nachbedingung:

Der BPEL-Prozess wird hochgeladen, gespeichert und steht für die Prozesstransformation bereit.

Alternative Abläufe:

Keinen

Anwendungsfall 2: Rules hochladen

Aktoren:

Der Benutzer

Vorbedingung:

Der BPEL-Prozess ist bereits hochgeladen und das navigierte Subinterface „Rules Dokument Upload" wird angezeigt.

Regulärer Ablauf:

1. Der Benutzer kopiert den kompletten Quellcode des Rules Dokuments ins Textfenster.
2. Der Benutzer klickt den Button „submit the rules".

Nachbedingung:

Die Prozesstransformation wird durchgeführt und das Ergebnis wird in dem Subinterface „Transformation Result" angezeigt.

Alternative Abläufe:

1. Das Subinterface „Transformation Result" wird nach jeder Prozesstransformation angezeigt.
2. Der Benutzer klickt auf den Button „new rules copy", um das Subinterface „Rules Dokument Upload" zu erreichen.

Anwendungsfall 3: Rules Design

Aktoren:

Der Benutzer

Vorbedingung:

Das Subinterface „Rules Dokument Upload" in der Seite „View Transformer" wird angezeigt.

Regulärer Ablauf:

1. Der Benutzer klickt auf den Button „rules generator".
2. Das Editor-Interface „Rules Designer" wird angezeigt.
3. Der Benutzer editiert interaktiv das Rules Dokument durch das Klicken auf die Tasten.

Nachbedingung:

Das Rules Dokument wird interaktiv editiert, gespeichert und steht für die Prozesstransformation bereit.

Alternative Abläufe:

1. Der Benutzer klickt auf das Webinterface „Rules Designer".
2. Das Subinterface im „Rules Designer" wird angezeigt und der Benutzer klick auf den Button „Rule Design!".
3. Das Editor-Interface „Rules Designer" wird angezeigt.
4. Der Benutzer editiert interaktiv das Rules Dokument durch das Klicken auf die Tasten.

Anwendungsfall 4: View-Funktionen aufrufen

Aktoren:

Der Benutzer

Vorbedingung:

Der BPEL-Prozess ist bereits hochgeladen und das Subinterface „Rules Dokument Upload" wird angezeigt.

Regulärer Ablauf:

1. Der Benutzer klickt auf den Button „view funktion" und dann wird die verfügbare View-Funktion im Navigation-Subinterface angezeigt.
2. Der Benutzer wählt die Funktion aus und gibt die erforderlichen Daten ein.
3. Der Benutzer klickt auf den Button „submit".

Nachbedingung:

Die Prozesstransformation bzw. die View-Funktion wird durchgeführt und das Ergebnis wird im Subinterface „Transformation Result" angezeigt.

Alternative Abläufe:

1. Das Subinterface „Transformation Result" wird nach jeder Prozesstransformation angezeigt.
2. Der Benutzer klickt auf den Button „View functions", um die Subinterfaces „Call Focus Function" und „Call Remove Fragment Function" zu erreichen.

Anwendungsfall 5: Transformation ausführen

Aktoren:

Der Benutzer

Vorbedingung:

Der BPEL-Prozess ist bereits hochgeladen, das Rules Dokument bereits editiert und das Subinterface „Rules Designer" wird angezeigt.

Regulärer Ablauf:

1. Der Benutzer klickt auf den Button „rules submit and transform".

Nachbedingung:

Die Prozesstransformation wird durchgeführt und das Ergebnis wird in dem Subinterface „Transformation Result" angezeigt.

Alternative Abläufe:

1. Nach dem Hochladen des Rules Dokuments wird die Prozesstransformation automatisch durchgeführt.
2. Nach dem Hochladen der Parametrisierung im View-Funktionen-Interface wird die Prozesstransformation automatisch durchgeführt.

Anwendungsfall 6: Service WSDL hochladen

Aktoren:

Der Benutzer

Vorbedingung:

Das Webinterface „Services Manager" wird angezeigt.

Regulärer Ablauf:

1. Der Benutzer klickt auf den Weblink im Subinterface „Services Import".
2. Der Benutzer sucht das lokale WSDL Dokument für die generierte Webseite aus und klickt auf den Button „upload the wsdl file to server".
3. Der Benutzer klickt auf den Kommandolink „Services list update".

Nachbedingung:

Das lokale Rules-Dokument wird hochgeladen und die Services darin werden durchsucht und dann in der Servicetabelle aufgelistet.

Alternative Abläufe:

Keine

Anwendungsfall 7: Service auswählen

Aktoren:

Der Benutzer

Vorbedingung:

Das Webinterface „Services Manager" wird angezeigt und es stehen bereits die Services zur Verfügung.

Regulärer Ablauf:

1. Der Benutzer klickt auf den Kommandolink in dem Subinterface „Begin View Design", um eine neue Konstruktion zu initialisieren.
2. Der Benutzer klickt auf den Kommandolink „parameters" in dem Subinterface „Services List" und es wird eine neue Webseite generiert.
3. Der Benutzer gibt die entsprechenden Daten ein.
4. Der Benutzer klickt auf den Button „Save the parameter inputs".

Nachbedingung:

Der Services wurde ausgewählt und die eingegebene Daten werden gespeichert.

Alternative Abläufe:

Keine

Anwendungsfall 8: Services Sequenz Deployment

Aktoren:

Der Benutzer

Vorbedingung:

Das Webinterface „Services Manager" wird angezeigt und es ist bereits mindestens ein Service ausgewählt in dem Subinterface „ Execute the called services sequence".

Regulärer Ablauf:

1. Der Benutzer klickt auf den Kommandolink „Deploy the sequence process".

Nachbedingung:

Die Services-Sequenz wird als ein komplettes BPEL-Projekt umgesetzt und in den Apache ODE bereitgestellt, kompiliert und aktiviertet.

Alternative Abläufe:

Keine

Anwendungsfall 9: Services Sequenz ausführen

Aktoren:

Der Benutzer

Vorbedingung:

Das Webinterface „Services Manager" wird angezeigt und es wurde bereits eine assoziierte Services-Sequenz in Apache ODE deployt.

Regulärer Ablauf:

1. Der Benutzer checkt den Status des BPEL-Projekts durch den Klick auf den Kommandolink „ Check Status" im Subinterface „Status of the deployed process".
2. Wenn der Prozess in den Apache ODE bereits aktiviert ist, dann klickt der Benutzer klickt auf den Kommandolink „Invoke the active process".
3. Wenn es sich „in processing, please wait" meldet, dann checkt der Benutzer den Status in einen Moment später aus.

Nachbedingung:

Der Prozess bzw. die Services-Sequenz wurde durchgeführt und das Ergebnis wird in dem Subinterface „Result of services transformation" angezeigt.

Alternative Abläufe:

Keine

Anwendungsfall 10: Services Sequenz download

Aktoren:

Der Benutzer

Vorbedingung:

Das Webinterface „Services Manager" wird angezeigt und der identifizierte Prozess wurde bereits erfolgreich durchgeführt.

Regulärer Ablauf:

1. Der Benutzer klickt auf den Kommandolink „Deployed process download".

Nachbedingung:

Das in Apache ODE deployte BPEL-Projekt wird komplett heruntergeladen.

Alternative Abläufe:

Keine

Anwendungsfall 11: View Manager einloggen

Aktoren:

Der Administrator

Vorbedingung:

Das Login Interface wird angezeigt.

Regulärer Ablauf:

1. Der Administrator gibt den korrekten Benutzername und Password ein.
2. Der Administrator klickt auf den Button „submit".

Nachbedingung:

Das Webinterface „Process View Manager" wird angezeigt.

Alternative Abläufe:

Keine

Anwendungsfall 12: View Manager ausloggen

Aktoren:

Der Administrator

Vorbedingung:

Der Administrator ist erfolgreich eingeloggt und das Webinterface „Process View Manager" wird angezeigt.

Regulärer Ablauf:

1. Der Administrator klickt auf den Button „Logout".

Nachbedingung:

Der Administrator ist erfolgreich ausgeloggt und das Login Webinterface wird angezeigt.

Alternative Abläufe:

Keine

Anwendungsfall 13: Transformation Item lesen

Aktoren:

Der Administrator

Vorbedingung:

Der Administrator ist erfolgreich eingeloggt und das Tabelleninterface „Process View Manager" wird angezeigt.

Regulärer Ablauf:

1. Der Administrator klickt den Weblink „show details" in der entsprechend gewünschten Zeile in den Tabellen an.

Nachbedingung:

Die assoziierte Informationen wie der BPEL-Prozess, das Rules Dokument und das View Ergebnis werden in einem neuen Fenster angezeigt.

Alternative Abläufe:

Keine

Anwendungsfall 14: Transformation Item löschen

Aktoren:

Der Administrator

Vorbedingung:

Der Administrator ist erfolgreich eingeloggt und das Tabelleninterface „Process View Manager" wird angezeigt.

Regulärer Ablauf:

1. Der Administrator klickt den Button „delete" in die entsprechenden Zeile in der Tabellen an.
2. Der Administrator bestätigt diesen Vorgang durch den Klick auf „OK".

Nachbedingung:

Diese registrierte Transformation wird in den Tabellen gelöscht.

Alternative Abläufe:

1. Der Administrator klickt den Button „Remove", um alle registrierten Transformationen in den Tabellen zu vernichten (Ein kompletter Datenverlust muss manuell vorgesorgt werden) .
2. Der Administrator bestätigt diesen Vorgang durch den Klick auf „OK".

3.2.2. Transformation Services Architektur

In der Arbeit wird die Realisierung und Verwaltung des Prozesstransformation-Webservices als die Kernaufgabe und Vertiefungsthema gerichtet. Es wird darin betrachtet und getestet, wie groß die realistische Anwendbarkeit und technische Durchführbarkeit sein wird, wenn man die entwickelte Prozesstransformations-Technologie auf das Webservices-Konzept umsetzten will. Es wird die bereits funktionierende View-Funktionen als aufrufbaren Webservices mit entsprechender Parametrisierung implementiert. Die Abbildung 3.3 zeigt eine schematische Architektur für die Verwaltung und Anwendung der elementaren Prozesstransformation, die in der Arbeit erstellt werden. Jeder Webservice für die Transformation des Prozessmodells wird in eine zentralen Servicebasis registriert.

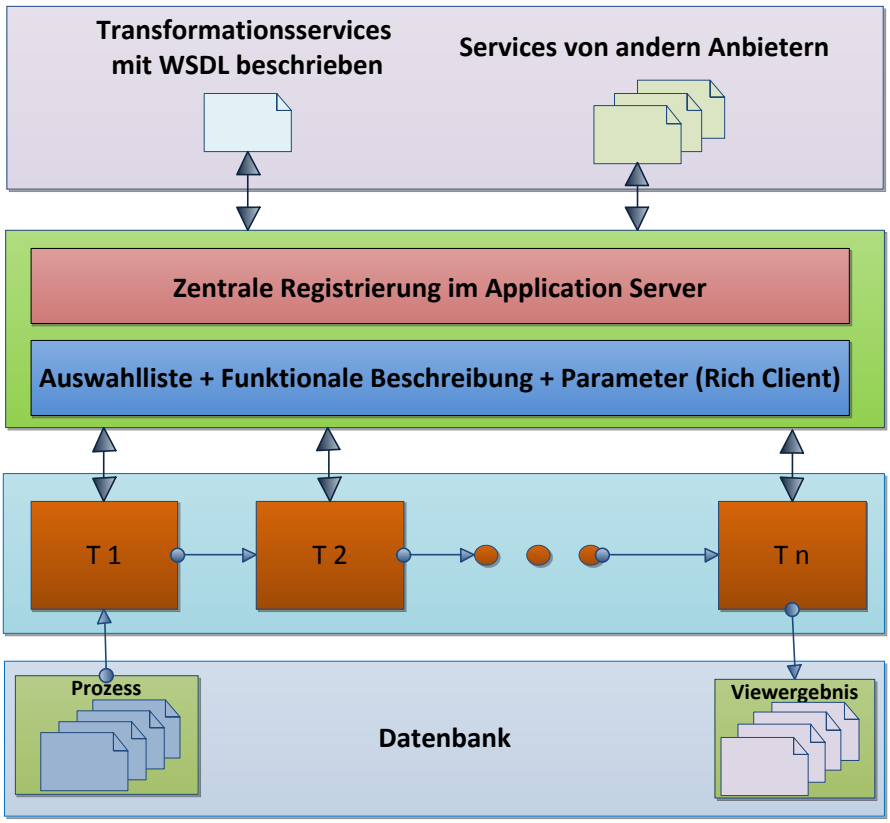

Abbildung 3.3.: Serviceorientierte Business Prozess Transformation Services Architektur

Um die unterstützte Funktionalität für die Prozesstransformation in der Verwaltungsplattform zu erweitern, wird ein Mechanismus unterstützt, mit dem mögliche Services für Prozess View aus anderen Organisation oder Partnern in diese Plattform integriert werden können. Das verfügbare externe WSDL-Dokument wird manuell in die Servicebase importiert. Der

implementierte ausführbare Transformation Services wird in allen WSDL-Dokumenten in der Servicebase sowie die Services Registrierung in der Anwendung werden sorgfältig durchsucht und in die Auswahllist hinzugefügt. Ein komfortables User-Interface für eine flexible Konstruktion bzw. Kombination einer Services-Sequenz wird in den weiteren Abschnitten 3.2.3 entworfen und spezifiziert. Der Benutzer kann mehrere Prozesstransformationen sequenziell kombinieren, um eine komplizierte Sicht auf den BPEL-Prozess zu erzeugen. Es wird ein weiterer Mechanismus bzw. Lösung herausgegeben, mit dem die Ausführung dieser definierte Transformation-Reihenfolge gewährleistet und effizienter implementiert wird. Er soll automatisch die Zielprozess Dokumente in der Datenbank abholen und in der benutzerdefinierten Reihenfolge transformieren. Die Schritte „T1", „T2" bis „Tn" in der Abbildung 3.3 sind die vom Benutzer ausgewählten Prozess Transformation Ablaufschritte, die sequenziell selbstständig in dem Verwaltungsplattform-Backend ausgeführt werden sollen.

In dem Sequenzdiagramm 3.4 werden die Benutzerinteraktion und die Prozessabläufe für die Serviceanwendung in der Verwaltungsplattform spezifiziert. Es wird darin jeder erforderlicher Verarbeitungsschritt für eine sequenzielle Konstruktion einer Sicht auf BPEL-Prozesse detailliert illustriert.

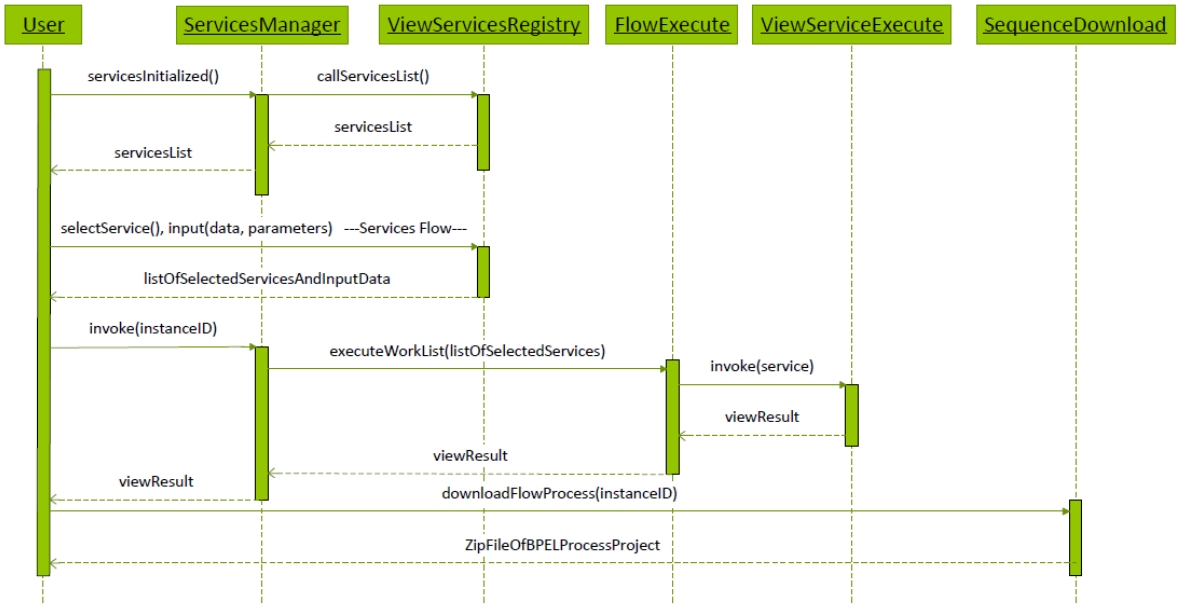

Abbildung 3.4.: Sequenzdiagramm von der Transformation Services Anwendung

3.2.3. Transformation Services Client (View Designer)

Die Kernaufgabe in dieser Arbeit ist die Entwicklung eines Prozess View Designer, die das Webservices-basierte Kombinieren von Prozesstransformationen sequentiell realisiert. In den vorherigen Abschnitten ist die Gesamtarchitektur für die Implementierung dargestellt. Um

diese Verwaltungsplattform für den View Designer zu aufbauen, wurden in den Entwurfsphase der praktische Arbeit vier Arbeitspakten wie folgt spezifiziert. Die für die relevanten Verarbeitungsschritten in Transformation Services Client verantwortlich sind.

Services Auflisten

Die Auflistung aller verfügbaren Prozesstransformation-Services für die Benutzer ist der erste Arbeitsschritt der Verwaltungsplattform. Die in der Arbeit erstellten Services bzw. die Operationen für Prozess View werden in WSDL Dokumenten beschrieben und sind bereits in das Service-Register bzw. in einem Ordner gespeichert. Ein Service-Finder durchsucht die unterstützten Services und deren unterliegenden Operationen sowie die entsprechende Parametrisierung in den WSDL Dokumenten und listet sie in eine Service-Tabelle für den Benutzern zur Verfügung auf. Zur Addition eines neuen Prozess View Service für die Verwaltungsplattform soll es optional durch das Hochladen eines WSDL Dokuments realisiert werden.

Services Kombination

Die in der Tabelle aufgelisteten Services und die dazu gehörenden Operationen können ausgewählt werden, um die implementierte Aktion für Prozess View zu anwenden. Es soll in eine weiteres Dateninterface nach die Auswahl übergeführt werden, in dem die entsprechenden Daten wie der Prozess, die Rules und die Parametern für die Operationen eingeben werden können. In der Praxis wird in einer Prozesstransformation nicht nur eine einzelne Operation aufgerufen. Die Benutzern können mehrere Aktionen sequenziell verbinden, um eine komplexe Transformation zu erzeugen. Es sollte darin dieser Mechanismus unterstützt werden, mit dem die nacheinander liegenden Operationen sequenziell ohne Datenverlust verknüpft werden. Ein flexibles Auswechseln zwischen den Operationen hilft eine erforderliche Modifikation in einem kleinen Teile der Sequenz. Die erzeugte Kombination und die eingegebenen Daten sollen eindeutig für die spätere Verarbeitung assoziiert werden, d.h, alle Informationen für jede Kombinationsinstanz werden in einem eindeutig (z.B, durch einen Zeitstempel) identifiziertem XML-Dokument strukturell gespeichert.

Services Deployment

Nach dem erfolgreichen Kombinieren des Services und der Dateneingabe resultiert identifizierbares XML-Dokument. In diesem Schritt soll ein BPEL-Projekt erzeugt werden, die diese Services Kombination implementieren kann. In der Praxisarbeit soll automatisch ein eindeutiger Ordner unter dem Apache ODE Dokumentpfad »WEB-INF/processes/« erstellt werden, welcher ein vollständiges BPEL-Projekt ist. Ein BPEL-Projekt beinhaltet den BEPL-Prozess, das WSDL Dokument von sich selbst, die weiteren beteiligten WSDL Dokumente des Prozess View Services und das deploy.xml Dokument. Neben dem Kopieren von den beteiligten WSDL Dokumenten ins BPEL-Projekt werden die anderen drei XML-Dokument

wie der eigentliche BPEL-Prozess, das eigene WSDL Dokument und deploy.xml automatisch aus dem identifizierten XML-Dokument generiert. Der Apache ODE Server checkt den neu erstellten Ordner und kompiliert ihn. Nach einem erfolgreichen und fehlerfreien Kompilieren ist der BPEL-Prozess aktiviert und bereitgestellt.

Services Ausführen

Vor dem Ausführen des Services wird ein Statuscheck für die Instanz durchgeführt. Der Benutzer fragt dabei über die Management-API von Apache ODE den Prozessstatus ab, ob der gewünschte Prozess bereits aktiviert ist. Nach einer positiven Abstimmung wird ein Aufruf des aktivierten BPEL-Prozess implementiert. Die assoziierten Daten für den Aufruf wird aus dem obigen identifizierten XML-Dokument abgeholt. Es generiert automatisch eine operativen SOAP Nachricht und schickt sie zu der Zugriffsadresse des Services ab. Nach einer sequenzielle Prozesstransformationen wird eine Antwort für den Benutzer erstellt und der verarbeitete XML-basierte BPEL-Prozess in dem User-Interface präsentiert. Ein Herunterladen des ganzen BPEL-Projekts wird unterstützt.

3.3. Business Process View Template

Business Process View Template besteht in dieser Arbeit für eine Prozess View Vorlage, die einen aus den aufgerufenen Transformation Services kombinierte Sequenz ist und selbst als wiederverwendbarer Webservices veröffentlicht werden kann. Eine Prozess View Vorlage realisiert eine benutzerdefinierte Sicht auf den Geschäftsprozess neben der entsprechenden Parametrisierung. In die Prozess View Vorlage wird ein spezifizierter Verarbeitungsprozess für das definierte Anwendungsziel und Abstraktionssinn beschrieben, die von dem konkreten Anwendungsszenario und der konkreten Geschäftsprozesseigenschaften abhängig sein können. Es soll auch implementierbar sein, dass man eine abstraktere Prozess View Vorlage für eine Geschäftsprozess Transformation für die verschiedene Geschäftsbranchen erstellen und anwenden will.

3.3.1. View Template Design

Vor der Konstruktion der Prozess View Vorlage wird zuerst der abstrahierte Verarbeitungsprozess sequenziell modelliert. Die in der Arbeit entwickelten View-Funktionen als Webservices können zusammengesetzt werden, um eine Vorlage für die benutzerdefinierte Sicht auf BPEL-Prozesse zu erstellen. Es können auch die grundlegenden Webservices verwendet werden, welche als ein einzelner Schritt in dem Verarbeitungsprozess betrachtet wird. Eine komplexe Vorlage kann aus anderen elementaren Prozess View Vorlagen zusammen komponiert werden. Eine Vorlage zu designen wird durchgeführt ohne eine Datenabgabe bzw. ohne das Hochladen des Prozesses. Dies wird immer stärker durch den entsprechende Kombinationskontext und Anwendungsziel bestimmt.

3.3.2. View Template Bereitstellung

Eine erstellte Prozess View Vorlage wird in den Apache ODE Server erfolgreich kompiliert und als aufrufbarer Webservice bereit gestellt. Um dieser Vorlage anzuwenden wird nur das WSDL Dokument in dem Vorlage-Projekt gebraucht. Das zu der Vorlage gehörende BPEL-Projekt wird heruntergeladen und das darin liegende WSDL Dokument von die Prozess View Vorlage wird zum Services Auflisten importiert, dann wird es als ein elementarer Service registriert.

3.3.3. View Template Anwenden

Die Anwendung der Prozess View Vorlage wird einfach durch Service Auswahl und die entsprechende Parametrisierung realisiert. Es wird in der Arbeit so implementiert, dass alle Geschäftsprozess-bezogenen Daten für die Prozesstransformation beim Aufrufen gemeinsam in dem User-Interface einmal eingetragen werden. In der folgende Abbildung 3.5 wird beispielsweise ein View Template angezeigt. Diese Prozess View Vorlage soll die existierenden spezifischen annotierten Aktivitäten in dem Prozess bereinigen. Die grüne Aktivitäten in den Prozess sind die speziell klassifizierten Tätigkeiten im individuellen Prozesskontext.

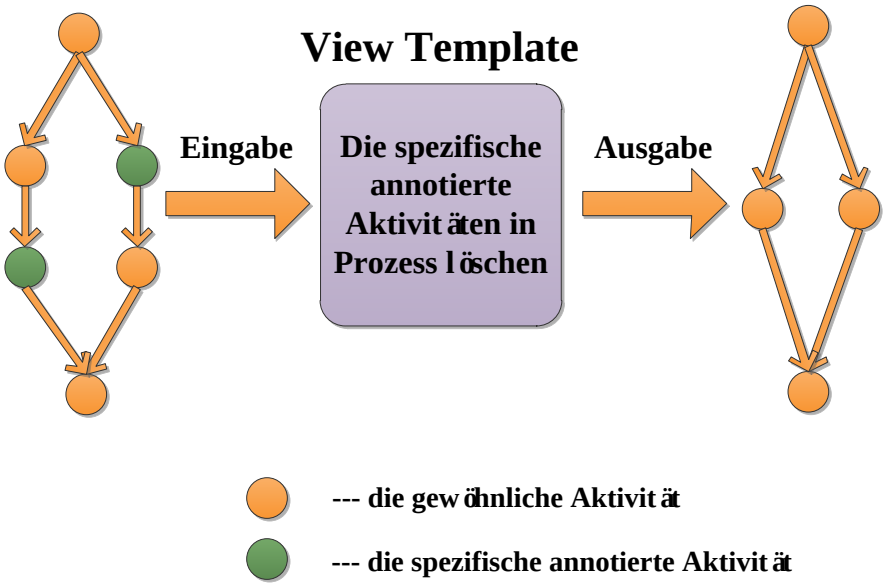

Abbildung 3.5.: Beispieldarstellung der View Template Anwendung

4. Implementierung

Im vorliegenden Kapitel wird die praktische Umsetzungen der im vorherigen Kapitel erklärten Konzepte und Entwürfen beschrieben. Es wird im ersten Abschnitt 4.1 die unterliegende Datenbankimplementierung erläutert. Im Abschnitt 4.2 wird das entworfene Webinterface und deren gegliederte Unterinterfaces für den Benutzer als eine webbasierte Verwaltungsplattform entwickelt. Das Kernkonzept der Arbeit, also die umgesetzten Transformation Services werden in dem Abschnitt 4.3 detailliert erläutert. Die Servicebasis für die technische Realisierung folgt in dem folgenden Abschnitt 4.4 bietet. Das neu erstellte Konzept der Prozess View Vorlage wird in dem Abschnitt 4.5 für eine praxisorientierte Anwendungsmöglichkeit vertiefter diskutiert.

4.1. Datenbanktabellen

Es wird in der Arbeit vorgenommen, alle ausgeführte Prozesstransformation in der Verwaltungsplattform zu protokollieren. Es werden darin die einfachen Datenschemen erstellt, um die assoziierten Daten, wie der BPEL-Prozess, das Rules Dokument, das Ergebnis und die Datumsinformation in eine relationale Datenbank, wie das hier angewendete MySQL detailliert und gut strukturiert zu speichern. In der Auflistung 4.1 werden die SQL-Ausdrücke für die Erzeugung solcher Datentabellen für die Verwaltungsplattform gezeigt. Es bestehen insgesamt drei Tabellen in der Datenbank, die die verschiedenen Anwendungsfunktionen für Datenobjektpersistenz realisieren.

Die Tabelle ***transformlist*** ist die zentrale Transformation Registrierung auf der Serverseite für die webbasierte Verwaltungsplattform. Es bietet vier Varianten für die Prozesstransformation in der Webanwendung. Die entsprechende Information der angewendeten Variante wird in der Spalte ***[info]*** registriert. Die Prozesstransformation durch Hochladen eines Rules-Dokuments wird es als vorgegebenes Verfahren „Default" protokolliert. Die Aufrufe der View-Funktionen wird als „Call view function" protokolliert. Das interaktive Editieren eines Rules-Dokuments wird als „Rules design" protokolliert und die Prozesstransformation durch die umgesetzten Transformation Services wird als „Call view service" protokolliert. Der Datentyp in den weiteren Spalten ***[process]***, ***[rules]***, ***[view]*** werden als ***LONGTEXT*** definiert, weil der Geschäftsprozess, das generierter Ergebnis und das Rules Dokument in den Praxis sehr groß sein können.

In der Tabelle ***servicesbean*** werden die verfügbaren Transformation Services für die Verwaltungsplattform gespeichert. Die in dem WSDL Dokument beschriebenen Services und die

detaillierte Parametrisierung werden ausgezogen und in den assoziierten Spalten dauerhaft registriert. Die relevanten Daten, wie Name und Datentyp jedes Parameters für die Operation werden in der Spalte ***[parameters]*** für das Services Auswählen textuell gespeichert.

Listing 4.1 Die SQL-Ausdrücke für die Erzeugung der Datenbanktabellen

```
create database processViewData;

use processViewData;

CREATE TABLE `processViewData`.`transformlist` (
  `transform_ID` INTEGER NOT NULL AUTO_INCREMENT,
  `info` LONGTEXT,
  `process` LONGTEXT,
  `rules` LONGTEXT,
  `view` LONGTEXT,
  `date` TEXT,
  PRIMARY KEY USING BTREE(`transform_ID`)
);

CREATE TABLE `processViewData`.`servicesbean` (
  `number` INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  `name` LONGTEXT,
  `operation` LONGTEXT,
  `output` LONGTEXT,
  `parameters` LONGTEXT,
  `request` LONGTEXT,
  `response` LONGTEXT,
  PRIMARY KEY USING BTREE(`number`)
);

CREATE TABLE `processViewData`.`parametersbean` (
  `id` INTEGER NOT NULL AUTO_INCREMENT,
  `datatyp` LONGTEXT,
  `instance` LONGTEXT,
  `operation` LONGTEXT,
  `previousServiceAsInput` TINYINT,
  `request` LONGTEXT,
  `response` LONGTEXT,
  `service` LONGTEXT,
  `value` LONGTEXT,
  PRIMARY KEY (`id`)
);
```

Die Tabelle ***parametersbean*** ist ein zeitweiliger Speicherort für die eingegebenen Daten in der Webservicekonzept-basierten Verwaltungsplattform. Die darin gespeicherten Daten bzw. die aufgerufene Operation und die eingegebenen Parameterwerten für jede eindeutige Konstruktionsinstanz werden in ein entsprechend identifiziertes XML-Dokument strukturiert transformiert. Die assoziierte Daten in dieser Tabelle werden dann wieder gelöscht. Es ist auch für den unabhängigen gleichzeitig Zugriff der verteilten Benutzer nacheinander gesorgt.

Die in der Datenbank benötigen Tabellen werden in der Anwendung als persistente Entity Typen durch den Hibernate EntityManager API Version 3.5 zusammen mit den Hibernate Annotations verwaltet und operiertet. Diese API beinhaltet einen Operationsset für die transaktionale Manipulation auf der persistenten Datenobjekte, z.B das Erstellen einer neuen persistenten Objektinstanz, das Modifizieren Objektinstanz, das Entfernen und das Durchführen einer Objektanfrage.

4.2. Web Client

In diesem Abschnitt wird die praktische Entwicklung eines webbasierten Benutzerinterface und die logische Funktionen in der Anwendungsschicht erläutert. Das Benutzerinterface in der Verwaltungsplattform ist in vier Webinterfaces unterteilt. Eine detaillierte Erklärung für jedes gegliederte Webinterface wird im Folgenden schrittweise gegeben.

4.2.1. View Services Manager

Das View Services Manager Webinterface ist das zentralen User-Interface der Verwaltungsplattform. Es wird wiederum in die verschiedenen Subfenstern bzw. Subfunktionen unterteilt. Die komplette Benutzerfunktionen werden sich innerhalb dieses Workbereich befinden. Das Webinterface ist für den Benutzer anschaulich und selbst erklärbar. Bevor die Anwendung gestartet wird, ist eine Initialisierung für eine neue Designsession erforderlich. In der Abbildung 4.1 ist die Designsession in dem View Services Manager schematisch dargestellt.

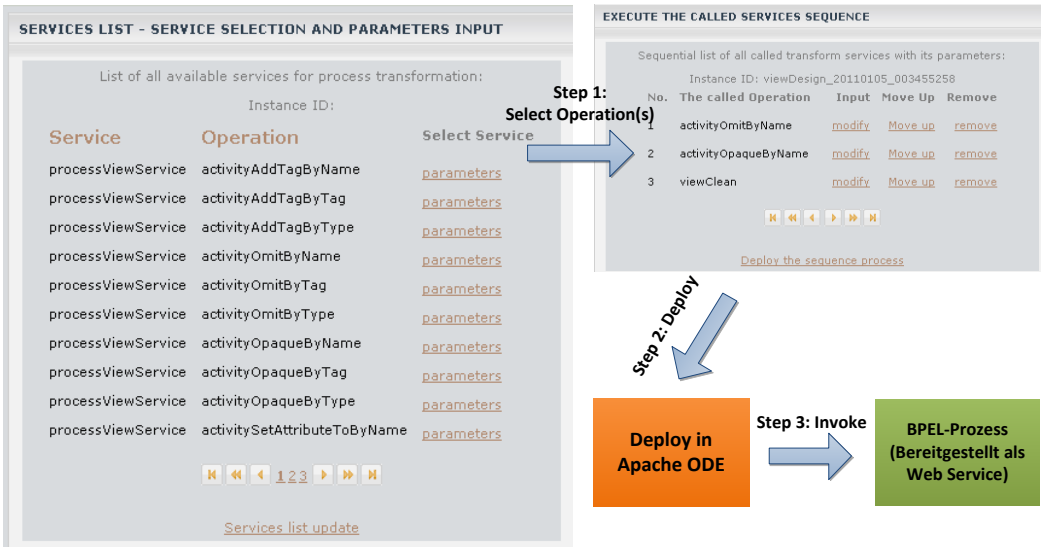

Abbildung 4.1.: Schematische Darstellung vom View Services Manager

Eine tabellarische Auflistung der verfügbare View Services sowie die rechte Tabelle „SERVICES LIST" in der obigen Abbildung 4.1 befinden sich im Mittelbereich des Webclients. Beim Rendering der JSP-Seite werden die Datensätzen, wie die umgesetzten Services und die darin unterstützten Operationen bzw. die entsprechenden Parametern aus der Datenbanktabelle ***servicesbean*** abgeholt und in dem Subfenster für die Anzeige der Servicebasis aufgelistet. Durch den Klick auf den Kommandolink „parameters" in der Zeile der Servicelist wird das Webinterface für die Datenabgaben des ausgewählten Services generiert und navigiert.

Nach jedem Service Auswählen und der Datenabgaben werden der aufgerufene Service in dem unteren Subfenster für die Anzeige der ausgewählten Service-Sequenz hinzugefügt und sequenziell aufgelistet, wie die linke Tabelle in der Abbildung 4.1. Es wird dabei ein flexibler Positionswechsel und Modifizierung der Servicedaten unterstützt. Durch den Klick auf den Kommandolink „Deploy the sequence process" wird die Sequenz in ein kompaktes BPEL-Projekt umgewandelt und in den Apache ODE auf der Serverseite bereitgestellt.

In dem links positionierten, kleinen Fenster für den Statuscheck des BPEL-Prozesses werden zwei Kommandolinks angeboten. Der Benutzer klickt auf den Kommandolink „Check Status", um den aktuellen Zustand des Prozesses zu befragen. Dabei wird in manchen Fälle z.B, bei zu vielen Benutzerabfragen, der deployte Prozess langsamer verarbeitet und verzögert. Nach einem aktivem Signal klickt der Benutzer auf den Kommandolink „Invoke the activ process" und die Durchführung der Prozesstransformation wird automatisch erzeugt.

Das Ergebnis wird dann in dem Textfenster textuell angezeigt und das ganze assoziierte BPEL-Projekt wird zum Herunterladen bereitgestellt. Ein Screenshot vom View Services Manager Webinterface wird in der Abbildung A.4 im Anhang gefunden. Die detaillierte Erklärung der grundlegenden Implementierungstechnik und die logischen Funktionen für das View Services Manager Webinterface werden in dem Abschnitt 4.4 schrittweise weiter diskutiert.

4.2.2. View Transformator

Ein View Transformator Webinterface soll die Arbeit der Durchführung einer Prozesstransformation erleichtern. Es ersetzt den Aufwand bei der Anwendung des durch Java implementierten konsolenbasierten Programmpaket. Eine Prozesstransformation wird relativ einfach durchgeführt durch das Hochladen der beiden beteiligten Dokumente, also des BPEL-Prozess und des Rules Dokument. Es werden darin auch die zwei bereits bestehenden View-Funktionen Fokus-Funktion und Teilprozesseliminierung aus der Arbeit [Cai10] implementiert.

4.2.3. Rules Designer

Ein interaktives webbasiertes Editor-Interface wird erstellt für das flexible Erzeugen eines Rules Dokument. Der Benutzer kann das Rules Dokument individuell durch das Klicken

auf die Tasten komfortabel erstellen. In dem Benutzerfenster werden der vordefinierte Funktionsbutton wie folgt bereitgestellt. Nach der Initialisierung eines Rules Dokuments wird das Wurzelelement „<tns:rules>" und die entsprechenden Attributen wie in die Auflistung 4.2 anzeigt automatisch vor erzeugt.

Listing 4.2 Das vorerst initialisierte Rules Dokument

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:rules xmlns:tns="http://www.eclipse.org/bpel/views/rules"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.eclipse.org/bpel/views/rules Rules.xsd"
    name="rulesGenerator">
</tns:rules>
```

Parametrisierung

Es wird zu erst die erforderliche allgemeine Parametrisierung für das Aggregat und die Bereinigung des Prozesses nach der Prozesstransformation spezifiziert. Nach der Selektion des entsprechenden booleschen Wertes werden die Angaben durch den Klick auf den Button „set parameters" bestätigt.

Rule hinzufügen

Durch den Klick auf den Button „add a rule" wird eine Anweisung neben dem eingetipptem eindeutigem Name in das Rules Dokument erstellt. Es wird ein funktionales aber unvollständiges Dokument (Ohne eine genaue Angabe der gewünschten Aktionen und Zielnavigation bzw. Zielselektion) in der Auflistung 4.3 angezeigt. Danach werden die Parametrisierung und eine Testanweisung für die Prozesstransformation eingegebenen.

Listing 4.3 Das Rules Dokument nach der Parametrisierung und dem Hinzufügen einer Anweisung

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:rules xmlns:tns="http://www.eclipse.org/bpel/views/rules"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="rulesGenerator"
    xsi:schemaLocation="http://www.eclipse.org/bpel/views/rules Rules.xsd">
  <parameter>
    <aggregate value="true"/>
    <cleaning value="true"/>
  </parameter>
  <rule apply="true" name="test">
    <actions/>
    <targets/>
  </rule>
</tns:rules>
```

Rule entfernen

Eine Aktion für die Entfernung einer Anweisung wird nützlich, wenn man das erstellte Rules Dokument modifizieren will. Durch die Angabe eines bereits stehenden und eindeutig identifizierten Name wird darin die entsprechende Anweisung komplett ausgelöscht.

Rule wählen

Eine Aktion für die Selektion einer Anweisung ist auch benutzbar, wenn man nur ein paar Details in einer Anweisung modifizieren will. Durch die Angabe eines bereits bestehenden und eindeutig identifizierten Name wird darin die entsprechende Anweisung ausgewählt. Eine detaillierte Modifikation wird durch die folgende Aktion gestützt.

Aktion hinzufügen und entfernen

Die entworfen fünf Aktionen „<actionOmit>", „<actionOpaque>", „<actionSetAttributeTo>", „<addPreserve>" und „<addTag>" werden ausgewählt, um die erstellte Anweisung und deren Funktionalität zu vervollständigen. Durch den Klick auf die Tasten und die Angabe von den entsprechenden Parameterwerten können die Aktionen in die bestimmte Anweisung hinzugefügt oder entferntet werden.

Zielobjektselektion hinzufügen und entfernen

Die drei Basisverfahren zur Zielobjektnavigation „<tag>", „<attribute>" und „<type>" werden unterstützt. Durch den Klick auf die Tasten und die Angabe von den entsprechenden Parameterwerten können die Ausdrücke für die Zielobjektselektion zu einer bestimmten Anweisung sequenziell hinzugefügt oder entferntet werden. Um die rekursive logische Kombination durch „<and>", „<or>" und „<not>" zu implementieren, wurde ein Mechanismus entwickelt, welcher eindasen Aufmachen und Schließen eines logische Operator realisiert. Damit wird ein beliebig geschachtelter logischer Ausdruck korrekt und strukturiert erstellt. In der Auflistung 4.4 wird ein Testbeispiel eines Logikausdrucks angezeigt. Darin wird jeder logischer Operator mit einem eindeutigen Namen und der Status-Signalisierung gekennzeichnet. Eine neu hinzugefügte Zielobjektnavigation wird in den am tiefsten eröffneten Logikoperator bzw. in dieser Situation in den „<not>" Operator eingefügt. Von dieser Erweiterung wird die vorhandene Prozesstransformation nicht beeinflusst.

Listing 4.4 Die mehrfach geschachtelten Logikausdrücke in dem Rules Dokument

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:rules xmlns:tns="http://www.eclipse.org/bpel/views/rules"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="rulesGenerator"
    xsi:schemaLocation="http://www.eclipse.org/bpel/views/rules Rules.xsd">
  <parameter>
    <aggregate value="true"/>
    <cleaning value="true"/>
  </parameter>
  <rule apply="true" name="test">
    <actions>
      <actionOmit/>
    </actions>
    <targets>
      <and id="1" status="open">
        <tag tagName="Stuttgart"/>
        <or id="2" status="closed">
          <tag tagName="Ulm"/>
          <tag tagName="Heidelberg"/>
        </or>
        <not id="3" status="open">
          <tag tagName="Mannheim"/>
        </not>
      </and>
    </targets>
  </rule>
</tns:rules>
```

4.2.4. View Administrator

In der Verwaltungsplattform wird ein User-Interface für den Administrator erstellt. Darin werden alle gespeicherten Prozesstransformationen auf dem Datenbanktabelle ***transformlist*** in den Webclient aufgelistet. Der Administrator kann die detaillierten Informationen über den abgegebenen BPEL-Prozess, das Rules Dokument und die resultierte Prozesssicht von jeder Transformation lesen und analysieren. Um diese Funktion in dem Interface zu realisieren, wird dafür das JSF 2.0 Frameworks angewendet. Eine Kollektion ***transformationList*** vom managt JavaBean ***transformationBean*** wird beim Rendering der JSP-Seite alle gespeicherten Instanzen durch eine Getter Methode sammeln. Durch die Anwendung der JSF-Tag ***dataTable*** werden alle abgeholten Datensätzen aus der Kollektion tabellarisch angezeigt. Das Tag-Element ***dataScroller*** in der JSF Komponente Tomahawk aus dem Apache MyFaces Projekt unterstützt eine Navigation in Datenkomponenten bzw. in einer Datentabelle durch Scrollen. Die genaue Informationen wird dann durch den navigierten Kommandolink „Show Details" abgefragt.

4.3. Prozess Transformation Service

In diesem Abschnitt werden der in der Arbeit implementierte Transformation Service für das BPEL-basierte Prozessmodell vorgestellt. Die technischen Details jeder Operation werden schematische erklärt und alle umgesetzten Transformation Services sollen eine Funktionsbase für die Verwaltungsplattform schaffen.

4.3.1. Operationen

Es wird in der Arbeit die grundlegenden Elementaroperationen und die bereits entwickelte View-Funktionen als verfügbare Webservices für die Prozesstransformation umgesetzt. In den Tabellen 4.1 und 4.2 werden alle aufrufbaren Operation aufgelistet und das entsprechende Anwendungsszenario werden kurz zusammengefasst.

Operation	Beschreibung
activityAddTagByName	Tag-Information addieren nach dem Name
activityAddTagByTag	Tag-Information addieren nach der Tag-Infos
activityAddTagByType	Tag-Information addieren nach der Type
activityOmitByName	Aktivitätslöschung nach dem Name
activityOmitByTag	Aktivitätslöschung nach der Tag-Infos
activityOmitByType	Aktivitätslöschung nach der Type
activityOpaqueByName	Undurchsichtig machen nach dem Name
activityOpaqueByTag	Undurchsichtig machen nach der Tag-Infos
activityOpaqueByType	Undurchsichtig machen nach der Type
activitySetAttrbuteToByName	Attribute modifizieren nach dem Name
activitySetAttrbuteToByTag	Attribute modifizieren nach der Tag-Infos
activitySetAttrbuteToByType	Attribute modifizieren nach der Type
activitySetPreservedByName	Aufbewahrt machen nach dem Name
activitySetPreservedByTag	Aufbewahrt machen nach der Tag-Infos
activitySetPreservedByType	Aufbewahrt machen nach der Type
processInput	Prozess hochladen
viewClean	Der Ergebnisprozess bereinigen

Tabelle 4.1.: Die als Webservices implementierte Elementaroperationen

Operation	Beschreibung
processViewFocusOnActivty	Fokussieren in den Prozess
processViewRemoveFragment	Teilprozesseliminieren
transformByRules	Prozesstransformation zum gegebenen Rules Dokument

Tabelle 4.2.: Die als Webservices implementierte View-Funktionen

Eine logische Kombination durch „and", „or" und „not" für die Zielobjektselektion in der Operation werden im aktuellen Stand dieser Arbeit nicht unterstützt. Es besteht noch diese Erweiterungsmöglichkeit, um die Zielobjektnavigation in der Operation zu vervollständigen.

4.3.2. WSDL

Alle aufrufbaren Operationen im Transformation Service werden in einem WSDL Dokument beschrieben. Dieses Dokument wird erstellt z.B durch den Aufruf unter diesem Weblink [1] nach einer erfolgreichen Anwendungsinstallation auf dem Tomcat Webserver. Das WSDL Dokument ist eines der relevanten Artefakte für den kompletten Konstruktionsablauf in der Webservicekonzept-basierten Verwaltungsplattform.

4.4. Services Anwendung und Verwaltung

In diesem Abschnitt werden die technischen Implementierungsdetails und die grundlegenden Logikfunktionen für das View Services Manager Webinterface vorgestellt. Das Kerngebiet der Arbeit liegt bei der Implementierung des View Services Managers. Es richtet sich an einen Webservices-basiertes Anwendungsaspekt für die Transformation des Prozessmodelles

4.4.1. Bereitstellung von Services

Die in den vorherigen Abschnitten erklärten Services bzw. Operationen werden in der Verwaltungsplattform als eine zentrale Funktionsbasis bereitgestellt. Auf dem Anwendungsserver wird ein spezieller Ordner unter dem Name „importedWSDL" für die Lagerung des relevanten WSDL Dokumentes erstellt. Ein individuelles Hochladung eines gewünschten externen WSDL Dokuments wird optional durch den Kommandolink „Services Import" unterstützt. Es wird vorgenommen. Nach einer erfolgreich Installation wird das WSDL Dokument „processViewService.wsdl" initiativ automatisch in die Lagerung gespeichert. Beim erstmaligen Aufruf des View Services Managers ist die Servicelist des Webclients leer

[1] http://localhost:8080/processViewWebAnwendung/services/processViewService?wsdl

und besitzt keine verfügbaren Operationen in der entsprechenden assoziierten Datenbanktabelle ***servicesbean***. Druch Klick auf den Kommandolink „Services list update" wird das Durchsuchen von allen verfügbaren Services und darin liegenden Operationen durchgeführt. Es werden die Services und Operationen in der Datenbanktabelle vollständig gespeichert . Der durchgeführte Verarbeitungsablauf ist wie folgt gegliedert.

Schritt 1: Dokument finden

Die Methode ***servicesListUpdate()*** durchsucht die XML-basierten Dokumente in dem Ordner „importedWSDL" rekursiv. Wenn der Dokumentname mit „.wsdl" endet, wird die DOM-Baumstruktur in den Zwischenspeichern analysiert und extrahiert. Die späteren Arbeitsschritte basieren auf diesem DOM-Baum.

Schritt 2: Service finden

Die Methode ***findServiceName()*** wird jeweils nach der jeweils Erstellung des DOM-Baum ausgelöst. Sie zielt auf das XML-Element „<wsdl:service>" ab und holt den Wert aus dem Attribut „name". Der Servicename wird dann assoziiert mit der Spalte „name" in der Datenbanktabelle ***sevicesbean*** .

Schritt 3: Operation finden

Es wird in dem WSDl Dokument eine Reihe von Operationen beschrieben. Die Methode ***findeServiceOperation()*** durchsucht die allen XML-Elemente „<wsdl:operation>" innerhalb des Elements „<wsdl:portType>" und extrahiert den Wert aus dem Attribut „name". Der Operationsname wird in der Spalte „operation" der Tabelle gespeichert.

Schritt 4: Eingabeparametern finden

Nach jeder erfolgreichen Registrierung einer Operation müssen die entsprechenden Parametern herausgefunden werden. Unter dem Element „<wsdl:operation>" werden die Nachrichtenformate für die Eingabe und Ausgabe definiert. Die Methode passt die verfügbaren Elemente „<wsdl:message>" im WSDL Dokument mit den Werten in dem Attribut „message" in den Elementen „<wsdl:input>" an. In dem zusammengepassten Element „<wsdl:message>" werden das Unterelement „<wsdl:part>" und dessen Attribut „element" weiter analysiert. In dem WSDL Dokument werden die XML Schema Elemente für jeden Datentypen spezifiziert. Die Methode findet die Datenstruktur und den entsprechenden Datentyp für jeden Eingabeparameter heraus. Solche Informationen werden dann in der Spalte „parameters" der Datenbanktabelle ***sevicesbean*** textuell registriert.

Schritt 5: Ausgabeparametern finden

Die Datenstruktur und die vordefinierten Datentypen für die Ausgabeparameter in „<wsdl:output>" für jede Operation wird dann wie im oben beschriebenen Verfahren durchsucht. Die Information, wie Ausgabeparameter und Datentyp werden dann in der Spalte „output" gespeichert. Es wird in der Arbeit in den meisten Fällen die Zeichenketten „retrun (xs:string)" in der Spalte gespeichert. Neben den Parameterinformation werden die Nachrichtennamen, wie die Werte aus dem Attribut „message" der Elemente „<wsdl:input>" und „<wsdl:output>" auch genutzt. Sie werden in den Spalten „request" und „response" in der Tabelle für jede assoziierte Operation registriert. Solche Informationen werden unverzichtbar verlangt im späteren Anwendungskontext.

Schritt 6: Datensatz speichern

Um den Datensatz von jeder Operation in den Service zu speichern, wird in der Arbeit das Spring Framework und Hibernate Framwork zusammen angewendet. Es wird eine JavaBean „servicesBean" erstellt, in der alle relevanten Operationsinformationen als Eigenschaften definiert werden. Durch ein paar Setter-Methoden und die Funktion „EntityManager.persist()" wird der Datensatz in der Datenbanktabelle ***sevicesbean*** gespeichert. Nach frm alle beschriebenen Arbeitsschritte ausgeführt wurden, wird eine vollständige Datenbanktabelle erstellt und als Servicebase für das Webinterface „Service Auflisten" bereitgestellt.

4.4.2. Kombination von Services

Vor der Auswahl des bereitgestellten Services wird eine neue und eindeutige Identifizierung für die Konstruktion initialisiert. Der Benutzer klickt auf „Click here for initialize a new view design", dann wird eine Instanz-ID mit dem Name des „viewDesign" und mit einem exakten aktuellen Zeitstempel erzeugt. Nach einer sequenziellen Auswahl der Operation resultiert eine Services-Sequenz. In der Arbeit wird dieser Vorgang in einem identifizierbaren XML-Dokument unter dem Namen der entsprechenden Instanz-ID repräsentiert.

Operation wählen

Durch den Klick auf den Kommandolink „parameters" wird die entsprechende Operation ausgewählt und der in der Spalte „parameters" assoziierten Datensatz in der Tabelle ***sevicesbean*** wird abgeholt. Die gespeicherten Zeichenketten werden dann analysiert. Die Anwendung extrahiert die Parameter und erzeugt die Datensätzen in den Datenbanktabelle ***paramertsbean*** ohne die entsprechende Dateneintragung. Nach einer erfolgreiche Operationsselektion wird auf das Webinterface für die Datenabgaben navigiert.

Daten abgeben

Im Webinterface für die Datenabgabe wird eine Instanz-bezogene Tabelle wie in der Abbildung 4.2 angezeigt. In der nun die Spalten „datatyp", „value" und „previousServiceAsInput" aus der Tabelle ***paramertsbean*** dargestellt sind. Die anderen Spalten werden automatisch durch die assoziierte Information aus der Tabelle ***sevicesbean*** und der aktuellen Instanz-Session gefüllt.

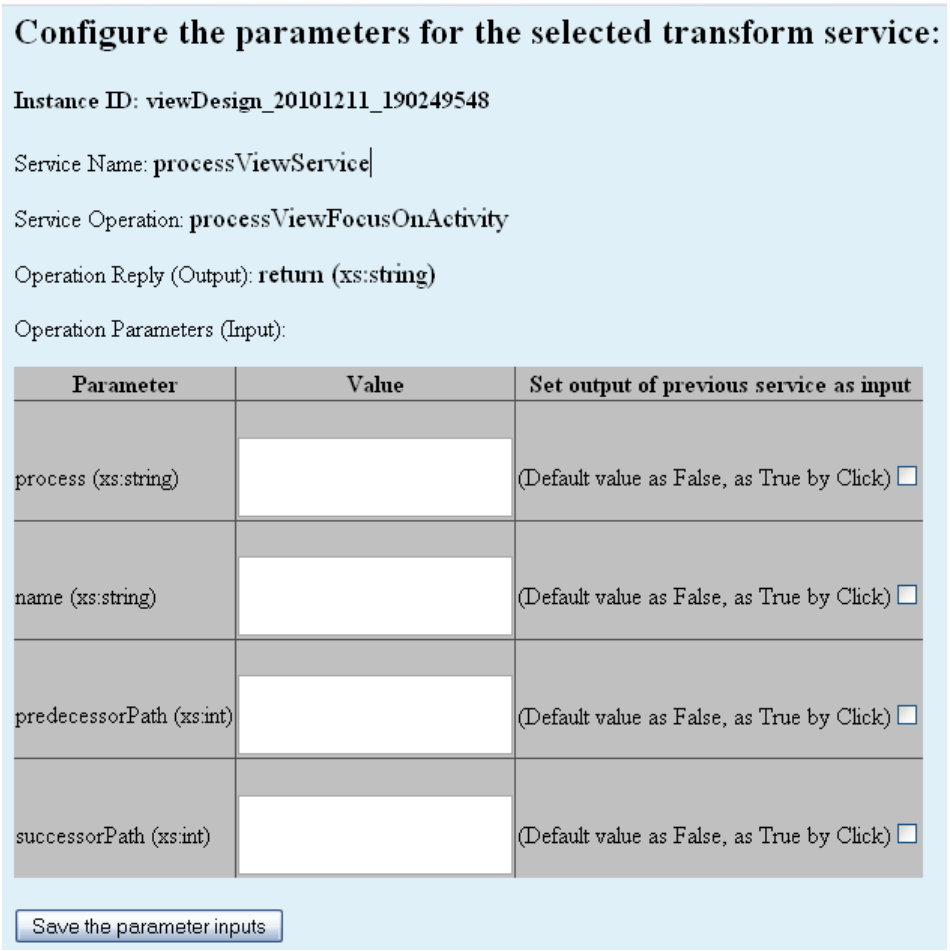

Abbildung 4.2.: Das Webinterface für die Datenabgabe

Es wird also angenommen, dass es bei der ersten Auswahl der Spalte „Set output of previous service as input" nicht selektiert wird muss. Diese Spalte dient als Kombinationspunkt zwischen den nacheinander ausgewählten Operationen. Eine boolesche Bewertung für jede folgende Operation ist dringend erforderlich, um den Ergebnisprozess der vorherigen Operation als eigenen Eingabeprozess zu bestimmen.

Zwischendokument erzeugen

Durch den Klick auf den Button „Save the parameter inputs" nach der Datenabgabe wird das Instanz-bezogene und eindeutig identifizierte XML-Dokument in dem spezialen Ordner „inputData" in der Verwaltungsplattform erzeugt. Die Anwendung liest die Instanz-bezogene Datensätzen aus der Tabelle ***paramertsbean*** und erzeugt das entsprechenden XML-basierte Dokument. Die bezogenen Datensätze werden nach der Erzeugung des XML Dokuments dann nicht weiter in der Tabelle ***paramertsbean*** gespeichert. In der Auflistung 4.5 wird ein Beispiel des erzeugten XML-Dokuments angezeigt.

Listing 4.5 Das erzeugte XML Dokument nach der Operationsauswahl und Datenabgabe

```
<input name="viewDesign_20101211_194951105">

  <service name="processViewService" operation="activityOmitByName" time="20101211_195346634">
    <request name="activityOmitByNameRequest">
      <process previous="false" type="string">
        <!-- DruckereiWorkflow BPEL Process [Generated by the Eclipse BPEL Designer] -->
        <!-- Date: Mon Nov 22 15:16:56 CET 2010 -->
        <bpel:process name="DruckereiWorkflow"
           targetNamespace="http://eclipse.org/bpel/sample"
           suppressJoinFailure="yes"
           xmlns:tns="http://eclipse.org/bpel/sample"
           xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">

                ---die detaillierte Prozessdefinition wird hier ausgelöscht.---

        </bpel:process>
      </process>
      <activityName previous="false" type="string">Cutting</activityName>
    </request>
    <response name="activityOmitByNameResponse"/>
  </service>

  <service name="processViewService" operation="activityOpaqueByName"
      time="20101211_195406863">
    <request name="activityOpaqueByNameRequest">
      <process previous="true" type="string"/>
      <activityName previous="false" type="string">Binding</activityName>
    </request>
    <response name="activityOpaqueByNameResponse"/>
  </service>

</input>
```

Zwischendokument modifizieren

Die in dem XML-Dokument repräsentierte Services-Sequenz wird in dem Webinterface „Services Sequenz" tabellarisch angezeigt. Es werden darin ein paar Funktionen unterstützt, um die eingelegte Sequenz und das assoziierte XML-Dokument leicht modifizieren und

anpassen zu können. Dazu gehört z.B die eingetragenen Daten zu verändern, die ausgewählte Operation zu löschen und die zwei nebeneinander positionierten Operationen auszuwechseln.

4.4.3. BPEL-Projekt Erzeugen und Deployment

In diesem Abschnitt wird die automatische Erzeugung des vollständigen BPEL-Projekts in der Apache ODE beschrieben. Alle folgenden Arbeitsschritte sollen nach dem Klick auf den Kommandolink „Deploy the sequenz process" nacheinander durchgeführt werden.

Schritt 1: BPEL Ordner erzeugen

Es wird zuerst ein Projektordner unter dem gleichen Name wie die eindeutig assoziierte Instanz-ID erzeugt. Dieser Ordner liegt in dem deployten Apache ODE Pfad, wie „ode/WEB-INF/processes". Das vorhanden WSDL Dokument „processViewService.wsdl" aus dem Ordner „importedWSDL" wird in den Projektordner kopiert. Die weiteren relevanten Artefakte für ein vollständiges BPEL-Projekt werden dann generiert.

Schritt 2: BPEL-Prozess erzeugen

Der BPEL-Prozess für jede eindeutige Konstruktionsinstanz wird zusammen durch das WSDL Dokument „processViewService.wsdl" und dem assoziierten Zwischendokument für die Service-Sequenz in den Ordner „inputData" automatisch produziert. Dieser Arbeitsvorgang wird in der Methode ***generateBPELFile()*** durchgeführt. Jede erforderliche Struktur, wie die XML-Elemente in dem BPEL-Prozess werden nacheinander in einem vordefinierten Prozessstil sequentiell generiert. In der Abbildung 4.3 ist der produzierte BPEL-Prozess für die definierte Sequenz in der Auflistung 4.5 grafisch dargestellt.

Der Mechanismus für die Erzeugung des BPEL-Prozesses und des WSDL Dokuments sind die Kernaufgabe in der Implementierungsphase. Das Implementierungskonzept basiert auf der Spezifikationen von WS-BPEL Version 2.0 und WSDL Version 1.1 bei der Erstellung solcher Artefakte. Die in dem Zwischendokument in Reihe definierten Operationsaufrufen werden in die BPEL-Struktur „<bpel:flow>" umgesetzt. Jede aufgerufene Operation wird als eine eindeutige identifizierte Aktivität „<bpel:invoke>" im BPEL-Prozess implementiert. Die Parameterübergabe wird durch die ebenfalls eindeutig identifizierte Aktivität „<bpel:assign>" realisiert.

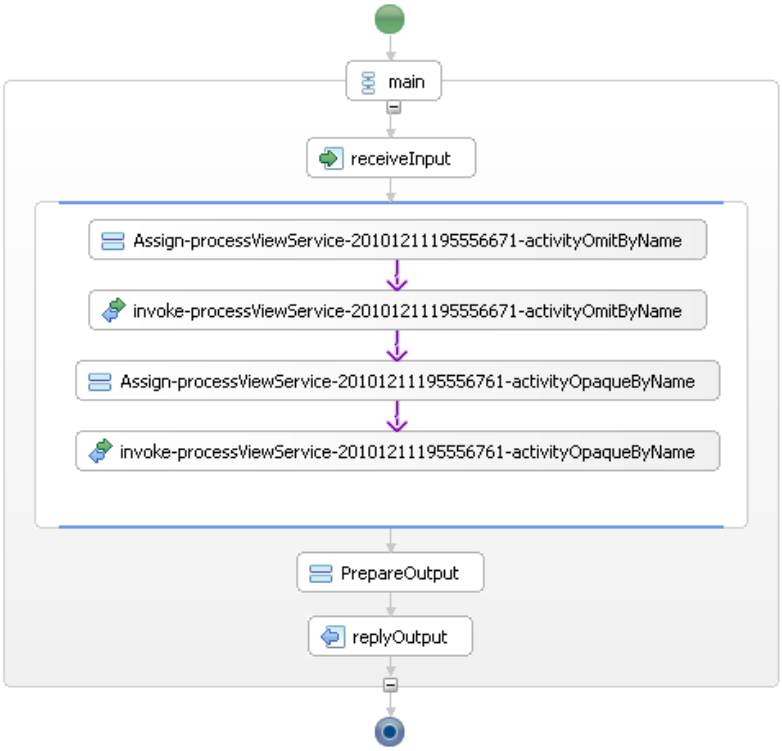

Abbildung 4.3.: Der erzeugter BPEL-Prozess für die Sequenz in der Auflistung 4.5

Schritt 3: WSDL Dokument erzeugen

Das spezifiziertes WSDL Dokument wird dann durch die Methode ***generateProcessWSDL()*** aus einem vollständigen BPEL-Prozess produziert. Das erzeugte WSDL Dokument für die Sequenz in der Auflistung 4.5 wird im Folgenden veranschaulicht.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:ns1="http://processView.com" xmlns:p="http://www.w3.org/2001/XMLSchema"
    xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
         xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.processView.com/viewDesign_20101211_194951105"
    name="viewDesign_20101211_194951105"
         targetNamespace="http://www.processView.com/viewDesign_20101211_194951105">
    <types>
        <schema xmlns="http://www.w3.org/2001/XMLSchema"
            attributeFormDefault="unqualified" elementFormDefault="qualified"
            targetNamespace="http://www.processView.com/viewDesign_20101211_194951105">
            <element name="viewDesign_20101211_194951105Request">
                <complexType>
                    <sequence>
```

```
                    <element minOccurs="0" name="process_20101211_195346634"
                        nillable="true" type="string" />
                    <element minOccurs="0" name="activityName_20101211_195346634"
                        nillable="true" type="string" />
                    <element minOccurs="0" name="activityName_20101211_195406863"
                        nillable="true" type="string" />
                </sequence>
            </complexType>
        </element>
        <element name="viewDesign_20101211_194951105Response">
            <complexType>
                <sequence>
                    <element minOccurs="0" name="result" nillable="true" type="string" />
                </sequence>
            </complexType>
        </element>
    </schema>
</types>
<import location="processViewService.wsdl" namespace="http://processView.com" />
<plnk:partnerLinkType
    name="processViewService-20101211195556671-activityOmitByNamePartnerLinkType">
    <plnk:role
        name="processViewService-20101211195556671-activityOmitByNameProvider"
        portType="ns1:processViewServicePortType">
    </plnk:role>
</plnk:partnerLinkType>
<plnk:partnerLinkType
    name="processViewService-20101211195556761-activityOpaqueByNamePartnerLinkType">
    <plnk:role
        name="processViewService-20101211195556761-activityOpaqueByNameProvider"
        portType="ns1:processViewServicePortType">
    </plnk:role>
</plnk:partnerLinkType>
<message name="viewDesign_20101211_194951105RequestMessage">
    <part element="tns:viewDesign_20101211_194951105Request" name="payload" />
</message>
<message name="viewDesign_20101211_194951105ResponseMessage">
    <part element="tns:viewDesign_20101211_194951105Response" name="payload" />
</message>
<message
    name="processViewService-20101211195556671-activityOmitByNameRequest">
    <part element="ns1:activityOmitByName" name="parameters" />
</message>
<message
    name="processViewService-20101211195556671-activityOmitByNameResponse">
    <part element="ns1:activityOmitByNameResponse" name="parameters" />
</message>
<message
    name="processViewService-20101211195556761-activityOpaqueByNameRequest">
    <part element="ns1:activityOpaqueByName" name="parameters" />
</message>
<message
    name="processViewService-20101211195556761-activityOpaqueByNameResponse">
    <part element="ns1:activityOpaqueByNameResponse" name="parameters" />
</message>
```

```
    <portType name="viewDesign_20101211_194951105">
        <operation name="process">
            <input message="tns:viewDesign_20101211_194951105RequestMessage" />
            <output message="tns:viewDesign_20101211_194951105ResponseMessage" />
        </operation>
    </portType>
    <plnk:partnerLinkType name="viewDesign_20101211_194951105">
        <plnk:role name="viewDesign_20101211_194951105Provider"
            portType="tns:viewDesign_20101211_194951105" />
    </plnk:partnerLinkType>
    <binding name="viewDesign_20101211_194951105Binding"
         type="tns:viewDesign_20101211_194951105">
        <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <operation name="process">
            <soap:operation soapAction="http://eclipse.org/bpel/sample/process" />
            <input>
                <soap:body use="literal" />
            </input>
            <output>
                <soap:body use="literal" />
            </output>
        </operation>
    </binding>
    <service name="viewDesign_20101211_194951105">
        <port binding="tns:viewDesign_20101211_194951105Binding"
             name="viewDesign_20101211_194951105Port">
            <soap:address
                location="http://localhost:8080/ode/processes/viewDesign_20101211_194951105" />
        </port>
    </service>
</definitions>
```

Schritt 4: deploy.xml erzeugen

In der Methode ***generateDeployXML()*** wird das XML Dokument „deploy.xml" automatisch erstellt. Die relevanten Elemente sind hier „<provide>", „<invoke>" und das entsprechende Unterelement „<service>".

Listing 4.6 Das erzeugte deploy.xml Dokument für die Sequenz in der Auflistung 4.5

```
<?xml version="1.0" encoding="UTF-8"?>
<deploy xmlns="http://www.apache.org/ode/schemas/dd/2007/03"
     xmlns:ns1="http://processView.com" xmlns:sample="http://eclipse.org/bpel/sample"
     xmlns:tns="http://www.processView.com/viewDesign_20101211_194951105">
   <process name="tns:viewDesign_20101211_194951105">
      <active>true</active>
      <retired>false</retired>
      <process-events generate="all"/>
      <provide partnerLink="client">
         <service name="tns:viewDesign_20101211_194951105"
              port="viewDesign_20101211_194951105Port"/>
      </provide>
      <invoke
           partnerLink="processViewService-20101211195556671-activityOmitByNamePartnerLink">
         <service name="ns1:processViewService" port="processViewServiceHttpSoap11Endpoint"/>
      </invoke>
      <invoke
           partnerLink="processViewService-20101211195556761-activityOpaqueByNamePartnerLink">
         <service name="ns1:processViewService" port="processViewServiceHttpSoap11Endpoint"/>
      </invoke>
   </process>
</deploy>
```

In diese Methode werden die assoziierten Servicenamen und Port in dem WSDL Dokument „processViewService.wsdl" herausgefunden. Die aufgerufenen „partnerLink" werden aus dem generierten BPEL-Prozess unter dem XML Element „<bpel:partnerLinks>" automatisch extrahiert.

Schritt 5: BPEL-Prozess kompilieren

Apache ODE erkennt das neue generierte BPEL-Projekt selbständig. Der ODE Server prüft den BPEL-Prozess syntaktisch und semantisch und kompiliert ihn. Es wird dann eine CBP-Datei in dem BPEL-Projekt erfolgreich erzeugt.

4.4.4. BPEL-Prozess Ausführen

In der letzte Phase wird der deployte BPEL-Prozess als Webservices aufgerufen. Die abgegebene Daten sowie der Geschäftsprozess und die Parameter werden vom Prozess Transformation Service verarbeitet. Natürlich werden die kompletten Arbeitsschritte voll automatisch auf der Serverseite durchgeführt. Der Benutzer muss nur den Status abfragen und klickt auf den Kommandolink „Invoke the active process", um diesen Vorgang auszulösen. Die relevante Arbeiten sind hier die Erzeugung einer absendbaren SOAP-Nachricht aus dem Zwischendokument und die Verarbeitung der Antwort. Dieser Abschnitt bezieht sich hauptsächlich auf die Implementierung eines komplexen Webservices-Klient.

Schritt 1: Status checken

Apache ODE bietet eine Prozess Management API an, welche die auf den deployten BPEL-Prozess und erzeugten Prozessinstanz bezogenen Managementfunktion unterstützt. Die API wird als einen Webservice Interface implementiert. Darin werden sechs Operation spezifiziert, mit denen deployte Prozesses und die Instanzen abgefragt und die Zustände im Apache ODE manipuliert werden können. Das WSDL Dokument der Prozess Management API kurz „pmapi.wsdl" können unter diesen Weblinks [2] gefunden werden. Es wird in der Arbeit die Operation „listProcesses" angewendet, um die detaillierte Information über einen spezifizierten BPEL-Prozess durch eine Prozessnamensfilterung abzufragen. Die erzeugte SOAP-Nachricht wird in der Auflistung 4.7 angezeigt.

Listing 4.7 Die erzeugte Abfragenachricht zur Apache ODE Prozess Management API

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
     xmlns:pmap="http://www.apache.org/ode/pmapi">
   <soap:Header/>
   <soap:Body>
      <pmap:listProcesses>
         <filter>name=viewDesign_20101212_160941250*</filter>
      </pmap:listProcesses>
   </soap:Body>
</soap:Envelope>
```

Die detaillierte Information vom deploytem BPEL-Prozesses sind in der folgenden Auflistung ausführlich dargestellt. Das Programm sucht die aktuelle Konstruktionsinstanz bzw. die angepasste Prozess-ID „<ns:pid>" heraus. Der Prozesszustand wird in dem Element „<ns:status>" angezeigt, z.B als ACTIVE signalisiert. Dieser komplette Serviceaufruf und die Antwortverarbeitung werden in der Methode ***statusCheck()*** implementiert.

```
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
   <soapenv:Body>
      <axis2ns2:listProcessesResponse xmlns:axis2ns2="http://www.apache.org/ode/pmapi">
         <process-info-list>
            <ns:process-info xmlns:ns="http://www.apache.org/ode/pmapi/types/2006/08/02/">
               <ns:pid>
    {http://www.processView.com/viewDesign_20101212_160941250}viewDesign_20101212_160941250-10
               </ns:pid>
               <ns:status>ACTIVE</ns:status>
               <ns:version>10</ns:version>
               <ns:definition-info>
                  <ns:process-name
                        xmlns:view="http://www.processView.com/viewDesign_20101212_160941250">
                     view:viewDesign_20101212_160941250
                  </ns:process-name>
               </ns:definition-info>
```

[2] http://svn.apache.org/repos/asf/ode/trunk/axis2/src/main/wsdl/pmapi.wsdl

```
<ns:deployment-info>
   <ns:package>viewDesign_20101212_160941250</ns:package>
   <ns:document>viewDesign_20101212_160941250.bpel</ns:document>
   <ns:deploy-date>2010-12-12T16:32:35.687+01:00</ns:deploy-date>
   <ns:deployer/>
</ns:deployment-info>
<ns:instance-summary>
   <ns:instances state="ACTIVE" count="0"/>
   <ns:instances state="COMPLETED" count="0"/>
   <ns:instances state="ERROR" count="0"/>
   <ns:instances state="FAILED" count="0"/>
   <ns:instances state="SUSPENDED" count="0"/>
   <ns:instances state="TERMINATED" count="0"/>
</ns:instance-summary>
<ns:properties/>
<ns:endpoints/>
<ns:documents>
   <ns:document>
      <ns:name>processViewService.wsdl</ns:name>
      <ns:type>http://schemas.xmlsoap.org/wsdl/</ns:type>
      <ns:source>file:/C:/Dokumente und
           Einstellungen/cai/Desktop/apache-tomcat-6.0.26/
webapps/ode/WEB-INF/processes/viewDesign_20101212_160941250/
processViewService.wsdl
      </ns:source>
   </ns:document>
   <ns:document>
      <ns:name>viewDesign_20101212_160941250.bpel</ns:name>
      <ns:type>http://schemas.xmlsoap.org/ws/2004/03/business-process/</ns:type>
      <ns:source>file:/C:/Dokumente und
           Einstellungen/cai/Desktop/apache-tomcat-6.0.26/
webapps/ode/WEB-INF/processes/viewDesign_20101212_160941250/
viewDesign_20101212_160941250.bpel
      </ns:source>
   </ns:document>
   <ns:document>
      <ns:name>viewDesign_20101212_160941250.cbp</ns:name>
      <ns:type>http://www.fivesight.com/schemas/2005/12/19/CompiledBPEL</ns:type>
      <ns:source>file:/C:/Dokumente und
           Einstellungen/cai/Desktop/apache-tomcat-6.0.26/
webapps/ode/WEB-INF/processes/viewDesign_20101212_160941250/
viewDesign_20101212_160941250.cbp
      </ns:source>
   </ns:document>
   <ns:document>
      <ns:name>viewDesign_20101212_160941250.wsdl</ns:name>
      <ns:type>http://schemas.xmlsoap.org/wsdl/</ns:type>
      <ns:source>file:/C:/Dokumente und
           Einstellungen/cai/Desktop/apache-tomcat-6.0.26/
webapps/ode/WEB-INF/processes/viewDesign_20101212_160941250/
viewDesign_20101212_160941250.wsdl
      </ns:source>
   </ns:document>
</ns:documents>
</ns:process-info>
```

```
        </process-info-list>
      </axis2ns2:listProcessesResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

Schritt 2: SOAP-Nachrichten erzeugen

Nach dem aktivierten Signal wird der BPEL-Prozess als aufrufbarer Webservices bereitgestellt. Der Benutzer klickt auf den Kommandolink „Invoke the active process" und löst diese Aktion aus. Das Programm erzeugt automatisch eine SOAP-Nachricht aus dem assoziierten Zwischendokument in dem Ordner „inputData". Das WSDL Dokument in dem BPEL-Projekt wird analysiert, die Variablen werden durchgesucht und als Input-Parameter in die SOAP-Nachricht hinzugefügt. Dann wird die entsprechende Datenabgabe aus dem Zwischendokument abgeholt und in die assoziierten Parametern als Wertzuweisung eingetragen.

Listing 4.8 Die erzeugte SOAP-Nachricht aus den Zwischendokument in der Auflistung 4.5

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
     xmlns:view="http://www.processView.com/viewDesign_20101211_194951105">
    <soap:Header/>
    <soap:Body>
        <view:viewDesign_20101211_194951105Request
             xmlns:view="http://www.processView.com/viewDesign_20101211_194951105">
            <view:process_20101211_195346634>
                <!-- DruckereiWorkflow BPEL Process [Generated by the Eclipse BPEL Designer]
                     -->
                <!-- Date: Mon Nov 22 15:16:56 CET 2010 -->
                <bpel:process name="DruckereiWorkflow"
                    targetNamespace="http://eclipse.org/bpel/sample"
                    suppressJoinFailure="yes"
                    xmlns:tns="http://eclipse.org/bpel/sample"
                    xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">

                        ---die detaillierte Prozessdefinition wird hier ausgelöscht.---

                </bpel:process>
            </view:process_20101211_195346634>
            <view:activityName_20101211_195346634>Cutting</view:activityName_20101211_195346634>
            <view:activityName_20101211_195406863>Binding</view:activityName_20101211_195406863>
        </view:viewDesign_20101211_194951105Request>
    </soap:Body>
</soap:Envelope>
```

Die obige SOAP-Nachricht wird automatisch zusammen aus dem WSDL Dokument in dem BPEL-Projekt und dem assoziierten Zwischendokument in der Auflistung 4.5 generiert. Das

Programm sendet diese SOAP-Nachricht zu dem BPEL-Prozess in der Apache ODE, in der er als ein aufrufbarer Web Services kompiliert und bereitgestellt wird.

Schritt 3: Antworten abholen

Nach der erfolgreichen Transformation soll der Ergebnisprozess zurück geschickt werden. Das Ergebnis in der SOAP-Antwort, wie in der Auflistung 4.9 wird dann extrahiert und in dem Webinterface angezeigt. Es wird in der Arbeit die Fehlermeldung auch für den Benutzer angezeigt, wenn der Fehler innerhalb der Prozesstransformation generiert wurde.

Listing 4.9 Die SOAP-Antwortnachricht zu der Auflistung 4.8

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <viewDesign_20101211_194951105Response
           xmlns="http://www.processView.com/viewDesign_20101211_194951105">
        <tns:result xmlns:tns="http://www.processView.com/viewDesign_20101211_194951105">

          <!-- DruckereiWorkflow BPEL Process [Generated by the Eclipse BPEL Designer] -->
          <!-- Date: Mon Nov 22 15:16:56 CET 2010 -->
          <bpel:process name="DruckereiWorkflow"
              targetNamespace="http://eclipse.org/bpel/sample"
              suppressJoinFailure="yes"
              xmlns:tns="http://eclipse.org/bpel/sample"
              xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">

           ---die detaillierte Prozessdefinition nach Transformation wird hier
                ausgelöscht.---

          </bpel:process>

        </tns:result>
      </viewDesign_20101211_194951105Response>
   </soapenv:Body>
</soapenv:Envelope>
```

Schritt 4: BPEL-Projekt herunterladen

Das ganzes BPEL-Projekt wird durch den Klick auf den Kommandolink „Deployed process download" heruntergeladen. Das Projekt und darin liegende Artefakte werden als ein komplettes ZIP-File komprimiert.

4.5. Business Process View Template

In diesem Abschnitt wird die Prozess View Vorlage für eine höhere Anwendbarkeit vertieft überlegt. Die Implementierbarkeit und technische Umsetzbarkeit werden ebenfalls getestet. Es wird aber nur eine grobe Idee in dieser Arbeit vorgestellt.

Es wurde in der Praxisarbeit diese Unterstützung zu der Prozess View Vorlage implementiert. Ein erstellter Prototyp der Prozess View Vorlage wird hier als einen benutzerdefinierte Eliminierung eines kritischen Bereichs oder Pfads in einem Geschäftsprozess angewendet, wie in der Abbildung 4.4 schematisch angezeigt. Es besteht eigentlich aus zwei Schritten in der Vorlage. In der Abbildung A.1 ist diese Vorlage als ein BPEL-Prozess grafisch dargestellt. Damit können der Geschäftsprozess in einem Massenverfahren zu einer bestimmten Zielsetzung hin transformiert werden. Der Benutzer gibt den Geschäftsprozess und die Parameter zu einer flexibler Definition eines kritischen Pfads für den ersten Schritt „Focus on a subprocess" ein. Der zweite Schritt „Remove the subprocess" wird folgend selbständig durchgeführt. Eine anschauliches Testbeispiel wird in der Abbildungen A.2 und A.3 angezeigt. Darin werden die geschäftlichen Tätigkeiten für die Definition eines Druckweiterverarbeitungsprozess in einer Druckerei, wie der rötlich eingefärbten Bereich ausgelöscht.

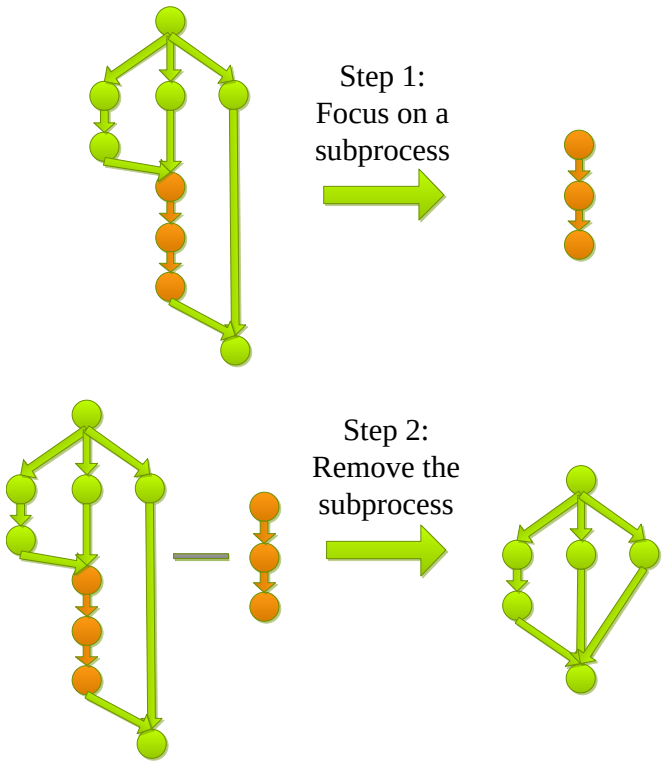

Abbildung 4.4.: Die grafische Darstellung dieser Prozess View Template und deren Prinzip

5. Zusammenfassung und Ausblick

In diesem Kapitel werden die Ergebnisse der Diplomarbeit kurz zusammengefasst und mit der primären Zielsetzungen verglichen. Ein Blick in die Zukunft, die bestehenden Verbesserungsmöglichkeiten und die betroffenen Probleme für die Business Prozess View werden ebenfalls betrachtet und im Kontext der aktuellen Rahmenbedingung und dem heutigen Technikstand für eine stärkere Anwendungskompetenz untersucht.

5.1. Zusammenfassung

Die Aufgabe in dieser Diplomarbeit ist die Entwicklung einer webbasierten Verwaltungsplattform für die benutzerdefinierte Prozesstransformation und die komfortable Konstruktion einer Sicht auf einen BPEL-Prozesses. Eine Applikation für die Prozesstransformation wurde bereits erstellt. Diese kann nach der Eingabe des BPEL-Prozesses und der XML-basierten Rulessprache über die Konsole das Prozessmodell transformieren. In der entworfenen Verwaltungsplattform wird diese Applikation integriert und ein benutzerfreundlicher Webclient dafür entwickelt, welcher in die vier Webinterface sowie View Designer und Manager, View Transformer, Rules Designer und View Administrator gegliedert ist. Die genutzten Frameworks für die Entwicklung einer Webanwendung in Mehrschichtenarchitektur werden in der Praxis umgesetzt und gegenseitig integriert.

Die elementarenn Operationen und die spezifizierten Funktionen für die Prozesstransformation werden in der Arbeit als aufrufbares Webservices Interface umgesetzt. Sie stellen eine zentrale Funktionsbasis für das View Designer und Manager dar. Damit ist die individuelle Konstruktion der Prozesstransformation implementierbar. Der Benutzer wählt die verfügbaren Operationen und kombiniert sie nacheinander. Die resultierende Sequenz wird zu einem ausführbaren BPEL-Prozess umgewandelt. Dieser vordefinierter Verarbeitungsprozess mit den Konstruktionsinstanz-bezogene Daten wird auf der Serverseite automatisch durchgeführt und das vollständige BPEL-Projekt wird zum Herunterladen bereitgestellt.

Die Webservice-basierte Konstruktion der benutzerdefinierten Prozesstransformation ist das hauptsächliche Praxisergebnis der Diplomarbeit. Es wird in der Webapplikation als Kernmodul betreut und bietet parallel eine höhere Erweiterbarkeit. Extern entwickelte Prozess Transformation Services können in dieser Plattform durch Hochladen des entsprechendes WSDL Dokumentes integriert werden. Die vielfältig angebotenen Funktionalitäten der Servicebasis werden dadurch verstärkt. Die konstruierte Services-Sequenz kann als eine wiederverwendbare Prozess View Vorlage für eine spezifizierte Prozessabstraktion in Rahmen eines Massenverfahrens wieder in die Plattform importiert werden.

5.2. Ausblick

Business Prozess View zeigt bereits seine große Kompetenz in der Geschäftsprozesstransformation. Seine Fähigkeit zur Prozessabstraktion und zum Verbergen der kritischen Informationen, wie z.B der geschäftsdaten-bezogenem Aktivitätskontext oder die spezielle Geschäftsregelung beinhaltetende Prozesssemantik in dem simulierten Geschäftsprozesses wird weiterhin vervollständigt und ständig verbessert. Künftig soll die Anwendungsvision in dem Gebiet der Business Prozess View weiter skaliert werden. Das Konzept der Prozess View Vorlage wäre ein nützlicher Ausgangspunkt, in dem die umgesetzten einfachen Transformation Services strukturell zusammengesetzt werden und damit ein eigener wiederverwendbarer Verarbeitungsmechanismus auf den Geschäftsprozess erstellt wird. Eine betroffene Anforderung ist die technische Umsetzbarkeit solcher elementaren und effektiven Operationsservices als grundsätzliche Bauelemente für die funktional stärkere Prozess View Vorlage.

Daneben wäre die Ausführbarkeit des Ergebnisprozesses nach der Prozesstransformation eine ausstehende Aufgabe. Im Verlauf der Diplomarbeit wurde diese Anforderung sowie die Simulierbarkeit des abstrahierten Prozesses wegen der resultierenden Hochkomplexität vernachlässigt. Dies bezieht sich weder auf die Prozessabstraktion noch auf die entsprechenden assoziierten Geschäftsdaten. Eine neue Lösung sollte dazu erstellt und implementiert werden, um den abstrahierten Prozesses ebenfalls korrekt simulieren und analysieren zu können.

Literaturverzeichnis

[AAA+07] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guízar, N. Kartha, et al. Web services business process execution language version 2.0. *OASIS Standard*, 11, 2007. (Zitiert auf den Seiten 7 und 14)

[Ber10] E. Bernard. Hibernate Annotations Reference Guide 3.5.1-Final, 2010. URL `http://docs.jboss.org/hibernate/stable/annotations/reference/en/pdf/hibernate_reference.pdf`. (Zitiert auf Seite 26)

[BHM+04] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, D. Orchard. Web Services Architecture, W3C Working Group Note 11 February 2004. *World Wide Web Consortium, article available from: http://www. w3. org/TR/ws-arch*, 2004. (Zitiert auf Seite 10)

[BK09] E. Burns, R. Kitain. JavaServer Faces Specification. *Sun Microsystems Inc. Santa Clara*, 2009. (Zitiert auf Seite 23)

[Cai10] J. Cai. Abstrakte Sichten auf BPEL Prozesse. Studienarbeit: Universität Stuttgart, Institut für Architektur von Anwendungssystemen, 2010. URL `http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=STUD-2250&engl=0`. (Zitiert auf den Seiten 8, 10, 18, 19, 33, 34 und 52)

[GKE10] M. R. A.-E. B. Gavin King, Christian Bauer, S. Ebersole. Hibernate Reference Documentation 3.5.1-Final, 2010. URL `http://docs.jboss.org/hibernate/stable/core/reference/en/pdf/hibernate_reference.pdf`. (Zitiert auf den Seiten 4 und 25)

[GSVR94] M. Gaitanides, R. Scholz, A. Vrohlings, M. Raster. *Prozessmanagement: Konzepte, Umsetzungen und Erfahrungen des Reengineering*. C. Hanser, 1994. (Zitiert auf Seite 7)

[KBRL05] N. Kavantzas, D. Burdett, G. Ritzinger, Y. Lafon. Web services choreography description language version 1.0, w3c candidate recommendation. *World Wide Web Consortium*, pp. 10–20051109, 2005. (Zitiert auf Seite 14)

[Mor09] R. Mordani. Java Servlet Specification Version 3.0. *Sun Microsystems, Inc.*, 2009. (Zitiert auf Seite 20)

[OAS07] OASIS. Web services business process execution language version 2.0. *URL: http://docs. oasis-open. org/wsbpel/2.0/OS/wsbpel-v2. 0-OS. html*, 2007. (Zitiert auf Seite 10)

[PD09] M. R. K.-m. C. Pierre Delisle, Jan Luehe. JavaServer Pages Specification. Version 2.2. *Sun Microsystems, Inc.*, 2009. (Zitiert auf den Seiten 21 und 22)

[PP08] M. Papazoglou, M. Papazoglou. *Web services: principles and technology*. Addison-Wesley, 2008.

[RJ+10] K. D. Rod Johnson, Juergen Hoeller, et al. The Spring Framework-Reference Documentation 3.0, 2010. URL `http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/pdf/spring-framework-reference.pdf`. (Zitiert auf den Seiten 4, 27, 28 und 29)

[SLS10] D. Schumm, F. Leymann, A. Streule. Process Viewing Patterns. In *Proceedings of the 14th IEEE International EDOC Conference, EDOC 2010, 25-29 October 2010, Vitória, Brazil*, pp. 89–98. IEEE Computer Society, 2010. doi:10.1109/EDOC.2010.16. (Zitiert auf Seite 7)

[Str09] A. Streule. *Abstract Views on BPEL Processes*. Diplomarbeit, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, 2009. (Zitiert auf den Seiten 8, 10, 18 und 33)

[Tea05] J. W. S. P. Team. Streaming APIs for XML Parsers, August 2005. URL `http://java.sun.com/performance/reference/whitepapers/StAX-1_0.pdf`. (Zitiert auf Seite 31)

[WCL+05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2005. (Zitiert auf den Seiten 4, 10 und 11)

Alle URLs wurden zuletzt am 20.12.2010 geprüft.

A. Anhang

A.1. Prozess View Template - Testbeispiele

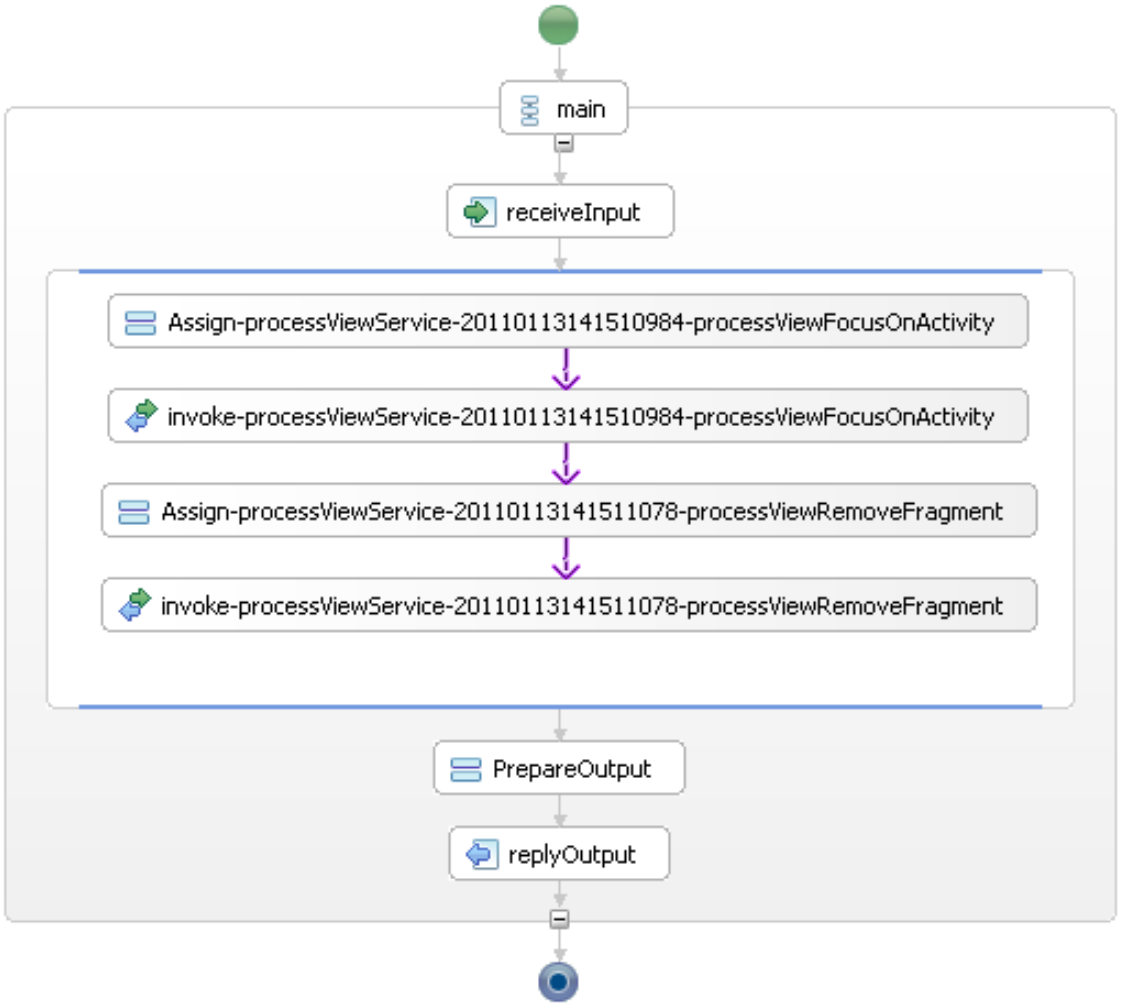

Abbildung A.1.: Prozess View Vorlage für die Elimination von dem PostPressManagement-Subprozess

Listing A.1 Das deploy.xml Dokument für die Prozess View Template in der Abbildung A.1

```
<?xml version="1.0" encoding="UTF-8"?>
<deploy xmlns="http://www.apache.org/ode/schemas/dd/2007/03"
    xmlns:ns1="http://processView.com" xmlns:sample="http://eclipse.org/bpel/sample"
    xmlns:tns="http://www.processView.com/viewDesign_20110113_141421125">
    <process name="tns:viewDesign_20110113_141421125">
        <active>true</active>
        <retired>false</retired>
        <process-events generate="all" />
        <provide partnerLink="client">
            <service name="tns:viewDesign_20110113_141421125"
                 port="viewDesign_20110113_141421125Port" />
        </provide>
        <invoke
            partnerLink="processViewService-20110113141510984-processViewFocusOnActivityPartnerLink">
            <service name="ns1:processViewService" port="processViewServiceHttpSoap11Endpoint"
                 />
        </invoke>
        <invoke
            partnerLink="processViewService-20110113141511078-processViewRemoveFragmentPartnerLink">
            <service name="ns1:processViewService" port="processViewServiceHttpSoap11Endpoint"
                 />
        </invoke>
    </process>
</deploy>
```

Der erzeugt entsprechend BPEL-Prozess für die Prozess View Template in der Abbildung A.1:

```
<?xml version="1.0" encoding="UTF-8"?>
<bpel:process xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
    xmlns:ns1="http://processView.com"
         xmlns:tns="http://www.processView.com/viewDesign_20110113_141421125"
    name="viewDesign_20110113_141421125" suppressJionFailure="yes"
    targetNamespace="http://www.processView.com/viewDesign_20110113_141421125">
    <bpel:import importType="http://schemas.xmlsoap.org/wsdl/"
        location="viewDesign_20110113_141421125.wsdl"
             namespace="http://processView.com/viewDesign_20110113_141421125" />
    <bpel:partnerLinks>
        <bpel:partnerLink myRole="viewDesign_20110113_141421125Provider"
            name="client" partnerLinkType="tns:viewDesign_20110113_141421125" />
        <bpel:partnerLink
            name="processViewService-20110113141510984-processViewFocusOnActivityPartnerLink"
    partnerLinkType="tns:processViewService-20110113141510984-processViewFocusOnActivityPartnerLinkType"
            partnerRole="processViewService-20110113141510984-processViewFocusOnActivityProvider"
            portType="ns1:processViewServicePortType" />
        <bpel:partnerLink
            name="processViewService-20110113141511078-processViewRemoveFragmentPartnerLink"
    partnerLinkType="tns:processViewService-20110113141511078-processViewRemoveFragmentPartnerLinkType"
            partnerRole="processViewService-20110113141511078-processViewRemoveFragmentProvider"
            portType="ns1:processViewServicePortType" />
```

```
    </bpel:partnerLinks>
    <bpel:variables>
        <bpel:variable messageType="tns:viewDesign_20110113_141421125RequestMessage"
            name="input" />
        <bpel:variable messageType="tns:viewDesign_20110113_141421125ResponseMessage"
            name="output" />
        <bpel:variable messageType="ns1:processViewFocusOnActivityRequest"
            name="processViewService-20110113141510984-processViewFocusOnActivityRequest" />
        <bpel:variable messageType="ns1:processViewFocusOnActivityResponse"
            name="processViewService-20110113141510984-processViewFocusOnActivityResponse" />
        <bpel:variable messageType="ns1:processViewRemoveFragmentRequest"
            name="processViewService-20110113141511078-processViewRemoveFragmentRequest" />
        <bpel:variable messageType="ns1:processViewRemoveFragmentResponse"
            name="processViewService-20110113141511078-processViewRemoveFragmentResponse" />
    </bpel:variables>
    <bpel:sequence name="main">
        <bpel:receive createInstance="yes" name="receiveInput"
            operation="process" partnerLink="client"
                 portType="tns:viewDesign_20110113_141421125"
            variable="input" />
        <bpel:flow name="Servicesflow">
            <bpel:links>
                <bpel:link name="link1" />
                <bpel:link name="link3" />
                <bpel:link name="link2" />
            </bpel:links>
            <bpel:assign
                name="Assign-processViewService-20110113141510984-processViewFocusOnActivity"
                validate="no">
                <bpel:copy>
                    <bpel:from>
                        <bpel:literal xml:space="preserve">
                            <proc:processViewFocusOnActivity
                            xmlns:proc="http://processView.com"
                                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                                <proc:process />
                                <proc:name />
                                <proc:predecessorPath />
                                <proc:successorPath />
                            </proc:processViewFocusOnActivity>
                        </bpel:literal>
                    </bpel:from>
                    <bpel:to part="parameters"
                        variable="processViewService-20110113141510984-processViewFocusOnActivityRequest"
                             />
                </bpel:copy>
                <bpel:copy>
                    <bpel:from part="payload" variable="input">
                        <bpel:query
                             queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
<![CDATA[tns:process_20110113_141459234]]></bpel:query>
                    </bpel:from>
                    <bpel:to part="parameters"
                        variable="processViewService-20110113141510984-processViewFocusOnActivityRequest">
```

```
                    <bpel:query
                        queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
<![CDATA[ns1:process]]></bpel:query>
                </bpel:to>
            </bpel:copy>
            <bpel:copy>
                <bpel:from part="payload" variable="input">
                    <bpel:query
                        queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
<![CDATA[tns:name_20110113_141459234]]></bpel:query>
                </bpel:from>
                <bpel:to part="parameters"
                    variable="processViewService-20110113141510984-processViewFocusOnActivityRequest">
                    <bpel:query
                        queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
<![CDATA[ns1:name]]></bpel:query>
                </bpel:to>
            </bpel:copy>
            <bpel:copy>
                <bpel:from part="payload" variable="input">
                    <bpel:query
                        queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
<![CDATA[tns:predecessorPath_20110113_141459234]]></bpel:query>
                </bpel:from>
                <bpel:to part="parameters"
                    variable="processViewService-20110113141510984-processViewFocusOnActivityRequest">
                    <bpel:query
                        queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
<![CDATA[ns1:predecessorPath]]></bpel:query>
                </bpel:to>
            </bpel:copy>
            <bpel:copy>
                <bpel:from part="payload" variable="input">
                    <bpel:query
                        queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
<![CDATA[tns:successorPath_20110113_141459234]]></bpel:query>
                </bpel:from>
                <bpel:to part="parameters"
                    variable="processViewService-20110113141510984-processViewFocusOnActivityRequest">
                    <bpel:query
                        queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
<![CDATA[ns1:successorPath]]></bpel:query>
                </bpel:to>
            </bpel:copy>
            <bpel:sources>
                <bpel:source linkName="link1" />
            </bpel:sources>
        </bpel:assign>
        <bpel:invoke
            inputVariable="processViewService-20110113141510984-processViewFocusOnActivityRequest"
            name="invoke-processViewService-20110113141510984-processViewFocusOnActivity"
            operation="processViewFocusOnActivity"
            outputVariable="processViewService-20110113141510984-processViewFocusOnActivityResponse"
            partnerLink="processViewService-20110113141510984-processViewFocusOnActivityPartnerLink"
            portType="ns1:processViewServicePortType">
```

```
                <bpel:targets>
                    <bpel:target linkName="link1" />
                </bpel:targets>
                <bpel:sources>
                    <bpel:source linkName="link2" />
                </bpel:sources>
            </bpel:invoke>
            <bpel:assign
                name="Assign-processViewService-20110113141511078-processViewRemoveFragment"
                validate="no">
                <bpel:copy>
                    <bpel:from>
                        <bpel:literal xml:space="preserve">
                        <proc:processViewRemoveFragment
                            xmlns:proc="http://processView.com"
                                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                            <proc:process />
                            <proc:fragment />
                        </proc:processViewRemoveFragment>
                        </bpel:literal>
                    </bpel:from>
                    <bpel:to part="parameters"
                        variable="processViewService-20110113141511078-processViewRemoveFragmentRequest"
                            />
                </bpel:copy>
                <bpel:copy>
                    <bpel:from part="payload" variable="input">
                        <bpel:query
                             queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
<![CDATA[tns:process_20110113_141508125]]></bpel:query>
                    </bpel:from>
                    <bpel:to part="parameters"
                        variable="processViewService-20110113141511078-processViewRemoveFragmentRequest">
                        <bpel:query
                             queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
<![CDATA[ns1:process]]></bpel:query>
                    </bpel:to>
                </bpel:copy>
                <bpel:copy>
                    <bpel:from part="parameters"
                        variable="processViewService-20110113141510984-processViewFocusOnActivityResponse">
                        <bpel:query
                             queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
<![CDATA[/*/text()]]></bpel:query>
                    </bpel:from>
                    <bpel:to part="parameters"
                        variable="processViewService-20110113141511078-processViewRemoveFragmentRequest">
                        <bpel:query
                             queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
<![CDATA[ns1:fragment]]></bpel:query>
                    </bpel:to>
                </bpel:copy>
                <bpel:targets>
                    <bpel:target linkName="link2" />
                </bpel:targets>
```

```
            <bpel:sources>
                <bpel:source linkName="link3" />
            </bpel:sources>
        </bpel:assign>
        <bpel:invoke
            inputVariable="processViewService-20110113141511078-processViewRemoveFragmentRequest"
            name="invoke-processViewService-20110113141511078-processViewRemoveFragment"
            operation="processViewRemoveFragment"
            outputVariable="processViewService-20110113141511078-processViewRemoveFragmentResponse"
            partnerLink="processViewService-20110113141511078-processViewRemoveFragmentPartnerLink"
            portType="ns1:processViewServicePortType">
            <bpel:targets>
                <bpel:target linkName="link3" />
            </bpel:targets>
        </bpel:invoke>
    </bpel:flow>
    <bpel:assign name="PrepareOutput" validate="no">
        <bpel:copy>
            <bpel:from>
                <bpel:literal xml:space="Preserve">
                    <tns:viewDesign_20110113_141421125Response
                        xmlns:tns="http://www.processView.com/viewDesign_20110113_141421125">
                        <tns:result />
                    </tns:viewDesign_20110113_141421125Response>
                </bpel:literal>
            </bpel:from>
            <bpel:to part="payload" variable="output" />
        </bpel:copy>
        <bpel:copy>
            <bpel:from part="parameters"
                variable="processViewService-20110113141511078-processViewRemoveFragmentResponse">
                <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
<![CDATA[/*/text()]]></bpel:query>
            </bpel:from>
            <bpel:to part="payload" variable="output">
                <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
<![CDATA[tns:result]]></bpel:query>
            </bpel:to>
        </bpel:copy>
    </bpel:assign>
    <bpel:reply name="replyOutput" operation="process"
        partnerLink="client" portType="tns:viewDesign_20110113_141421125"
        variable="output" />
</bpel:sequence>
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/"
    location="processViewService.wsdl" namespace="http://processView.com" />
</bpel:process>
```

Der erzeugt entsprechend WSDL Dokument für die Prozess View Template in der Abbildung A.1:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:ns1="http://processView.com" xmlns:p="http://www.w3.org/2001/XMLSchema"
```

```
xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://www.processView.com/viewDesign_20110113_141421125"
name="viewDesign_20110113_141421125"
     targetNamespace="http://www.processView.com/viewDesign_20110113_141421125">
<types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema"
       attributeFormDefault="unqualified" elementFormDefault="qualified"
       targetNamespace="http://www.processView.com/viewDesign_20110113_141421125">
       <element name="viewDesign_20110113_141421125Request">
           <complexType>
               <sequence>
                   <element minOccurs="0" name="process_20110113_141459234"
                       nillable="true" type="string" />
                   <element minOccurs="0" name="name_20110113_141459234"
                       nillable="true" type="string" />
                   <element minOccurs="0" name="predecessorPath_20110113_141459234"
                       nillable="true" type="int" />
                   <element minOccurs="0" name="successorPath_20110113_141459234"
                       nillable="true" type="int" />
                   <element minOccurs="0" name="process_20110113_141508125"
                       nillable="true" type="string" />
               </sequence>
           </complexType>
       </element>
       <element name="viewDesign_20110113_141421125Response">
           <complexType>
               <sequence>
                   <element minOccurs="0" name="result" nillable="true" type="string" />
               </sequence>
           </complexType>
       </element>
    </schema>
</types>
<import location="processViewService.wsdl" namespace="http://processView.com" />
<plnk:partnerLinkType
    name="processViewService-20110113141510984-processViewFocusOnActivityPartnerLinkType">
    <plnk:role
       name="processViewService-20110113141510984-processViewFocusOnActivityProvider"
       portType="ns1:processViewServicePortType">
    </plnk:role>
</plnk:partnerLinkType>
<plnk:partnerLinkType
    name="processViewService-20110113141511078-processViewRemoveFragmentPartnerLinkType">
    <plnk:role
       name="processViewService-20110113141511078-processViewRemoveFragmentProvider"
       portType="ns1:processViewServicePortType">
    </plnk:role>
</plnk:partnerLinkType>
<message name="viewDesign_20110113_141421125RequestMessage">
    <part element="tns:viewDesign_20110113_141421125Request" name="payload" />
</message>
<message name="viewDesign_20110113_141421125ResponseMessage">
    <part element="tns:viewDesign_20110113_141421125Response" name="payload" />
</message>
```

```
    <message
        name="processViewService-20110113141510984-processViewFocusOnActivityRequest">
        <part element="ns1:processViewFocusOnActivity" name="parameters" />
    </message>
    <message
        name="processViewService-20110113141510984-processViewFocusOnActivityResponse">
        <part element="ns1:processViewFocusOnActivityResponse" name="parameters" />
    </message>
    <message
        name="processViewService-20110113141511078-processViewRemoveFragmentRequest">
        <part element="ns1:processViewRemoveFragment" name="parameters" />
    </message>
    <message
        name="processViewService-20110113141511078-processViewRemoveFragmentResponse">
        <part element="ns1:processViewRemoveFragmentResponse" name="parameters" />
    </message>
    <portType name="viewDesign_20110113_141421125">
        <operation name="process">
            <input message="tns:viewDesign_20110113_141421125RequestMessage" />
            <output message="tns:viewDesign_20110113_141421125ResponseMessage" />
        </operation>
    </portType>
    <plnk:partnerLinkType name="viewDesign_20110113_141421125">
        <plnk:role name="viewDesign_20110113_141421125Provider"
            portType="tns:viewDesign_20110113_141421125" />
    </plnk:partnerLinkType>
    <binding name="viewDesign_20110113_141421125Binding"
         type="tns:viewDesign_20110113_141421125">
        <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <operation name="process">
            <soap:operation soapAction="http://eclipse.org/bpel/sample/process" />
            <input>
                <soap:body use="literal" />
            </input>
            <output>
                <soap:body use="literal" />
            </output>
        </operation>
    </binding>
    <service name="viewDesign_20110113_141421125">
        <port binding="tns:viewDesign_20110113_141421125Binding"
             name="viewDesign_20110113_141421125Port">
            <soap:address
                location="http://localhost:8080/ode/processes/viewDesign_20110113_141421125" />
        </port>
    </service>
</definitions>
```

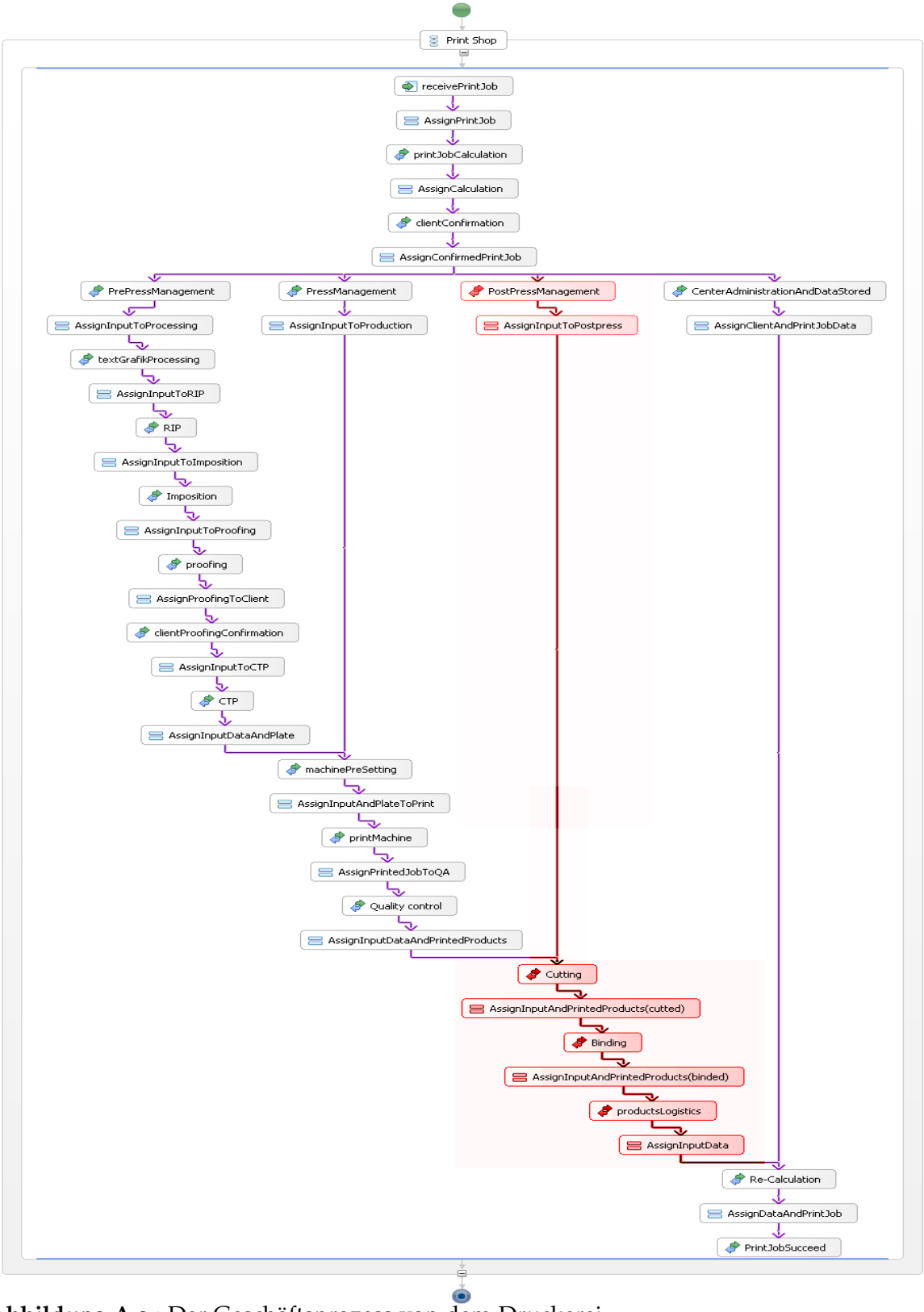

Abbildung A.2.: Der Geschäftsprozess von dem Druckerei

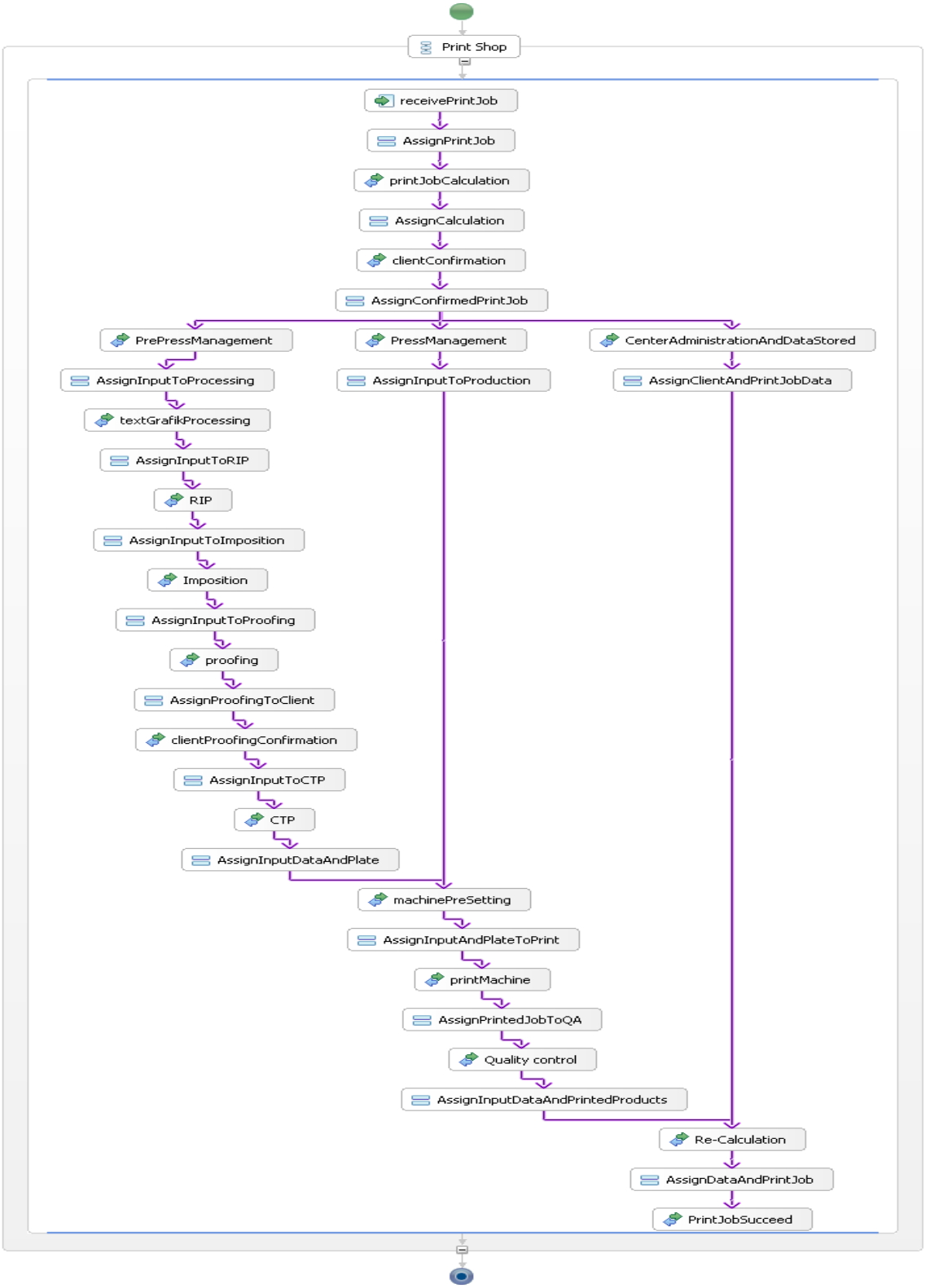

Abbildung A.3.: Der Geschäftsprozess von dem Druckerei nach die Elimination von dem PostPressManagement-Subprozess

A.2. Der Screenshot der Prozess View Verwaltungsplattform

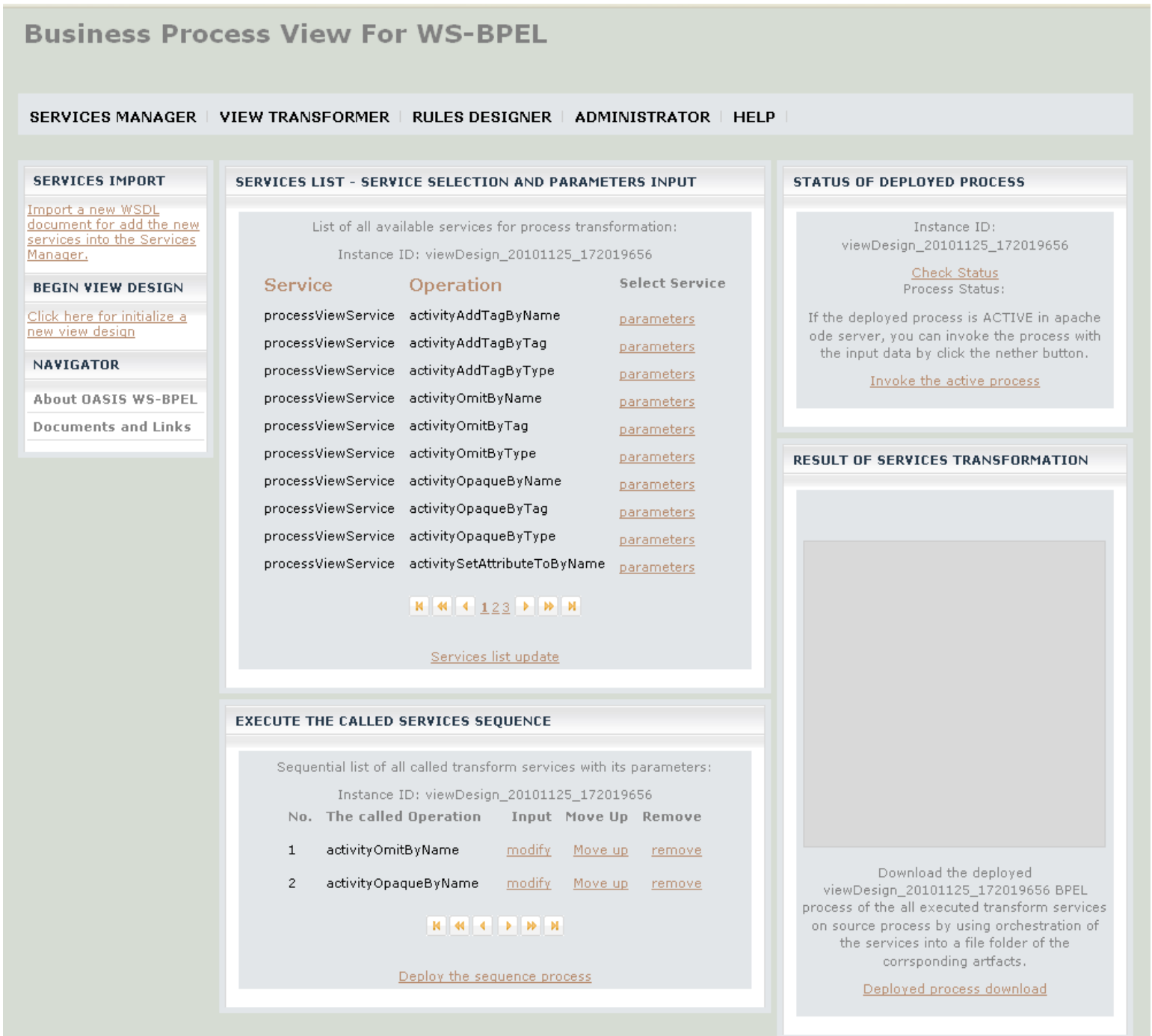

Abbildung A.4.: Prozess View Services Manager und Designer

Erklärung

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

(Jiayang Cai , am 20.12.2010)