Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart

Diplomarbeit Nr. 2852

Web Service Deployment Strategien

Robert Titze

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Frank Leymann
Betreuer:	Ralph Mietzner
begonnen am:	24.10.2008
beendet am:	24.04.2009
CR-Klassifikation:	D.2.11, D.2.13, H.3.5, H.4.1, K.6.3

Inhaltsverzeichnis

Vorwort ... 4
Kurzzusammenfassung ... 5
1. Einleitung ... 6
1.1. Einleitung in das Themengebiet ... 6
1.2. Motivation ... 6
1.3. Aufgabenstellung ... 6
1.4. Gliederung der Arbeit ... 7
2. Theoretische Grundlagen ... 8
2.1. SOA und Web Services ... 8
2.1.1. Services ... 8
2.1.2. Service-Orientierte Architekturen ... 9
2.1.3. Web Services ... 10
2.2. Workflow Management und BPEL ... 11
2.2.1. Business Process Management ... 11
2.2.2. Workflow Management ... 12
2.2.3. Business Process Execution Language ... 13
2.2.4. Service Component Architecture ... 15
2.3. SaaS und Variabilitäten ... 16
2.3.1. Software as a Service ... 16
2.3.2. eCCo als SaaS Anwendung ... 18
2.3.3. Multi-Tenancy-Patterns ... 18
2.3.4. Variabilitäten ... 21
3. Spezifikation ... 25
3.1. Anforderungen ... 25
3.1.1. Ziel und Zweck des Prototyps ... 25
3.1.2. Fachliches Sollkonzept ... 26
3.2. Darstellung der Anwendungsfälle ... 29
3.2.1. Akteure ... 29
3.2.2. Anwendungsfälle ... 30
3.3. Datenmodell ... 33
3.3.1. Kunden und Händler ... 34
3.3.2. Fahrzeuge und Sonderausstattungen ... 34
3.3.3. Dienstleistungen und Geschäftsdokumente ... 34
3.4. Spezifikation der fachlichen Prozesse ... 35
3.4.1. Vorbemerkungen ... 35
3.4.2. Angebotsprozess ... 37
3.4.3. Bestellprozess ... 39
3.4.4. Verwaltungsprozesse ... 41
3.5. Beschreibung der Variabilitäten ... 42
3.5.1. Fachliche Variabilitäten ... 42
3.5.2. Modellierung der Variabilitäten ... 43

4. Implementierung ... 46
4.1. Ablauf der Implementierung ... 46
4.1.1. Systemumgebung ... 46
4.1.2. Entwicklungsprozess ... 46
4.1.3. Architekturüberblick ... 47
4.2. Umsetzung des Geschäftsprozessmodells ... 47
4.2.1. Export des Geschäftsprozessmodells ... 47
4.2.2. Import der Workflows ... 49
4.3. Implementierung und Integration ... 52
4.3.1. Implementierung der Web Services ... 52
4.3.2. Anbindung der Datenbank ... 54
4.3.3. Anbindung der Benutzeroberfläche ... 56
4.3.4. Modulübersicht ... 60
5. Test der Anwendung ... 61
5.1. Testszenarien ... 61
5.2. Durchführung der Tests ... 61
5.2.1. Test der Worflows ... 61
5.2.2. Test der Web Services ... 63
5.2.3. Systemtest ... 65
5.3. Auswertung ... 68
6. Evaluierung der Ergebnisse ... 69
6.1. Deployment Strategien ... 69
6.1.1. Vorbemerkungen ... 69
6.1.2. Hinzufügen von Märkten und Händlern ... 69
6.1.3. Änderungen am Geschäftsmodell ... 70
6.2. Dienstgüteeigenschaften ... 70
6.2.1. Betrachtung der Anwendung ... 71
6.2.2. Einfluss der Variabilitäten ... 72
6.3. Fazit ... 72
7. Schlussbetrachtung und Ausblick ... 73
7.1. Schlussbetrachtung ... 73
7.2. Ausblick ... 74
Literaturverzeichnis ... 76
Abbildungsverzeichnis ... 81
Tabellenverzeichnis ... 82
Abkürzungsverzeichnis ... 83
Inhalte der CD-ROM ... 84
Erklärung ... 85

Vorwort

Diese Arbeit stellt den Abschluss meines Studiums dar, und beendet damit einen wichtigen Lebensabschnitt. Rückblickend möchte ich mich an dieser Stelle bei einigen Menschen bedanken, die mich auf meinem Weg begleitet haben.

Vielen Dank an Prof. Frank Leymann und seinen Mitarbeitern, die mein Interesse am Themengebiet Architektur von Anwendungssystemen geweckt haben. Danke an Fritz Fausel für die Unterstützung und die Bereitstellung einer angenehmen Arbeitsumgebung. Besonderer Dank an Heiko Kopp und Ralph Mietzner für die vorbildliche Betreuung meiner Diplomarbeit.

Für die Unterstützung danke ich meinen Kollegen Thomas Lee, Yves Maurischat, Roland Ramm und Rainer Wasilewski sowie Max Naguschewksi. Danke auch an Burkhard Weller und sein Team bei der Daimler AG für die Gelegenheit, erlernte Konzepte aus dem Studium praxisnah anwenden zu können.

Zu guter Letzt, ein herzliches Dankeschön an Anita und meine Familie, die immer an mich geglaubt haben. Schön, dass es Euch gibt!

Kurzzusammenfassung

Automobilhersteller und andere produzierende Unternehmen nutzen zahlreiche Softwaresysteme und Infrastrukturen. Diese sind im Laufe der Zeit gewachsen und nutzen unterschiedlichste Technologien, was die Kosten für Wartung und Integration ständig steigen lässt. Immer dynamischer werdende Märkte und veränderte Gesetze verlangen skalierbare Anwendungen, die zeitnah änderbar sind.

Service-Orientierte Architekturen und Web Service Technologien zerlegen Systeme in klar voneinander unterscheidbare Komponenten, die einzeln ausgetauscht werden können. Durch die Nutzung von Workflow Technologien werden diese Komponenten zu einem Ablauf zusammengesetzt, was die Integration mit Fremdsystemen erleichtert. Neue Ansätze, wie Software as a Service (SaaS) ermöglichen signifikante Verbesserungen im Bereich der Wartbarkeit und Skalierbarkeit von Anwendungen.

In der Forschung entwickelte Konzepte erleichtern die Beschreibung unterschiedlicher Softwarevarianten. Dabei werden variable Teile einer Anwendung getrennt denen beschrieben, die für alle gleich sind. Des Weiteren existieren Muster für Web Services, die die Nutzung selbiger durch mehrere Mandanten ermöglichen. Diese Arbeit evaluiert diese Konzepte anhand der Entwicklung einer Vertriebsanwendung für Autohändler.

1. Einleitung

1.1. Einleitung in das Themengebiet

Mit dem Ziel der Entwicklung einer Bestell-Anwendung für Autohändler wurde von der GFI iT-media Consult GmbH das Projekt „eCommerce Concept“ (eCCo) ins Leben gerufen. Die Software dient als Technologieprototyp, der zunächst für ein Drei-Schichtenmodell [FDF02] entworfen wurde, und deckt die Funktionalitäten Fahrzeugkonfiguration, Preisberechnung und Angebotserstellung ab.

1.2. Motivation

Heute setzen Autohersteller zahlreiche Softwaresysteme und Infrastrukturen [IBM1] ein, denen die unterschiedlichsten Technologien zugrunde liegen. Jede Geschäftseinheit verwaltet ihre eigenen Anwendungen, was einen erheblichen Integrationsaufwand zur Folge hat. Dies erschwert die Nutzung von Skaleneffekten, die einen klaren Wettbewerbsvorteil ermöglichen. Kurze Entwicklungszyklen und eine kontinuierliche Optimierung der Wartungskosten sind zu einer Kernanforderung geworden. Da in jedem Markt unterschiedliche Gesetze und Preise bestehen, müssen die eingesetzten Anwendungen hinreichend flexibel und wartbar sein. Neue Technologien ermöglichen die flexible Wiederverwendung einzelner Anwendungsteile auf Basis einer gemeinsamen Infrastruktur.

1.3. Aufgabenstellung

eCCo 5 ist auf Basis einer Service-Orientierten Architektur [WCL05] neu zu entwickeln, in der der IBM WebSphere Process Server 6.1 [IBM5] für die Ausführung der fachlichen Prozesse (engl.: workflow) zum Einsatz kommt. Mithilfe von Web Service Technologien [W3C1] kann die Software mit fremden Systemen, beispielsweise mit dem einer Versicherung, interagieren. Eine derartige Software ermöglicht es, Händlerprozesse zu vereinheitlichen und die Software-Wartungskosten zu minimieren. In jedem Markt kommen entweder unterschiedliche Prozessvarianten, oder aber unterschiedliche Web Services für die jeweiligen Berechnungen zum Einsatz. Die Anzahl der verschiedenen Prozesse und Web Services muss für eine effiziente Wartung gering gehalten werden. Ferner soll der Betrieb durch die Installation (engl.: deployment) der Workflows in neuen Märkten möglichst nicht gestört werden.

Am Institut für Architektur von Anwendungssystemen (IAAS) wurde eine Methodik entwickelt, bei der so genannte „Variabilitätsbeschreibungen“ [MLP08] verwendet werden, um Varianten in Workflows und den zugehörigen Aktivitäten zu beschreiben. Weiterhin existieren so genannte „Multi-Tenancy-Patterns“ [MUT09], mit denen einzelne Artefakte (z.B. Web Services) in einem „Software as a Service“ [CC06] Szenario genutzt werden können, um unterschiedliche Anforderungen an Wiederverwendung und Anpassbarkeit zu realisieren.

Ziel der Arbeit ist es, geeignete Deployment Strategien für die Prozesse der Angebotserstellung und Bestellabwicklung in verschiedenen Märkten zu erarbeiten. Dafür sind die am IAAS entwickelten Methoden anhand von repräsentativen Beispielprozessen und einem Prototyp zu evaluieren.

Die Handlungsvorschläge der Arbeit werden in den Folgeprojekten von eCCo 5 Berücksichtigung finden.

1.4. Gliederung der Arbeit

Die Gliederung der Arbeit orientiert sich an den einzelnen Entwicklungsphasen des Prototyps. Nach einer Einführung in das Themengebiet in Kapitel 2 werden die Anforderungen in Kapitel 3 spezifiziert. Kapitel 4 beschreibt die Implementierung und Kapitel 5 den Ablauf der Tests. Zum Abschluss werden die Ergebnisse in Kapitel 6 evaluiert und in Kapitel 7 zusammengefasst.

2. Theoretische Grundlagen

2.1. SOA und Web Services

2.1.1. Services

Getrieben durch geschäftliche und technologische Faktoren hat sich das Paradigma der „Service-Orientierten Architektur“ (SOA) innerhalb der letzten Jahre zu einer wichtigen Methode der IT-Welt entwickelt [WCL05]. Viele Firmen sehen sich heute gezwungen, möglichst schnell auf die Veränderungen hoch dynamischer Märkte zu reagieren. Dies benötigt vor allem flexibel anpassbare IT-Systeme, die es ermöglichen, sich ändernde Abläufe in kurzer Zeit umsetzen zu können.

Ziel produzierender Unternehmen ist es, Produkte an ihre Kunden zu verkaufen. Ein „Geschäftsprozess“ einer Firma beschreibt den Ablauf, wie diese Produkte hergestellt und vermarktet werden. Geschäftsprozesse sind eine Aneinanderreihung einzelner Aktivitäten, beispielsweise die Erfassung von Kundendaten vor einem Verkaufsgespräch. In einer SOA heißen solche Aktivitäten Dienste (engl.: services). Services benötigen Ressourcen, wie zum Beispiel ein Telefon oder eine Datenbank.

Aus technischer Sicht ist ein Service eine Software, die aufgerufen werden kann und einen bestimmten Zweck erfüllt. Setzt man Services zusammen, so beschreiben sie wiederum einen Ablauf, der auch „Orchestrierung“ genannt wird. Um dies zu ermöglichen, müssen Services „auffindbar“ sein und mithilfe einer Schnittstelle (auch: „Metadaten“) beschrieben werden. Eine Schnittstelle beschreibt einen Service vollständig, so dass keinerlei Kenntnis über den internen Ablauf für die Benutzung nötig ist. Damit wird ein Service zu einer Blackbox und gibt nach außen hin nur bekannt, wie er aufgerufen werden kann. Diese Idee erinnert an die Konzepte des „Information Hidings“ und der „Modularisierung“, die seit Jahrzehnten in der Informatik verbreitet sind [PAR72].

Weitere fundamentale Bestandteile einer SOA sind schon länger bekannt, wie das Konzept der „losen Kopplung“ einzelner Komponenten, welches eine Voraussetzung für die Verteilung von Softwaresystemen und für eine erfolgreiche Integration fremder Systeme ist. Neu hingegen ist es, den Anbieter eines Dienstes ebenfalls zu abstrahieren. Demjenigen, der den Service aufruft („Service Anforderer“), muss nicht bekannt sein, wer in anbietet. Dadurch werden nicht nur die dem Service zugrunde liegenden Implementierungen zur Laufzeit austauschbar („dynamisches Binden“), sondern es können ganze Systemteile an Geschäftspartner abgegeben werden.

2.1.2. Service-Orientierte Architekturen

Der Begriff „Service-Orientierte Architektur“ beschreibt einen Architekturstil für Softwaresysteme, der lose Kopplung und dynamisches Binden ermöglicht. Das Grundprinzip einer SOA ist in Abbildung 1 beschrieben.

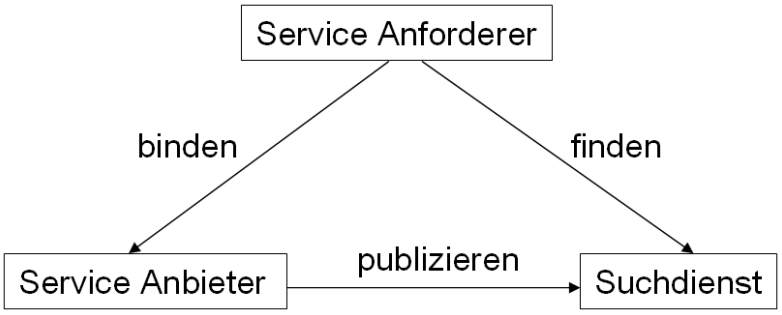

Abbildung 1: SOA-Dreieck [WCL05]

Der Service Anbieter beschreibt dabei zunächst seinen Service und sorgt dafür, dass er aufgerufen werden kann. Anschließend publiziert er ihn, d.h., er gibt ihn einem Suchdienst bekannt. Der Service Anforderer stellt seine Anfrage an den Suchdienst und wird an den zugehörigen Service „gebunden“ (engl.: binding). Nach dem Binden kann der Service Anforderer die vom Service angebotenen Operationen nutzen. Die Vorgänge Finden und Binden werden heute typischerweise von einem Enterprise Service Bus (ESB) [CHA04] übernommen.

Viele Anbieter von Softwareentwicklungsumgebungen empfehlen SOA als Grundlage für den Aufbau geschäftlicher Softwaresysteme. IBM [IJC07] teilt, wie in Abbildung 2 gezeigt, den Lebenszyklus eines solchen Softwaresystems in vier Phasen auf. Begonnen wird mit der Modellierung eines Geschäftsmodells, das die Ziele, Annahmen und Geschäftsprozesse des Unternehmens beinhaltet. Auf Basis dieses Geschäftsmodells werden die benötigten Services ermittelt. Letztere werden anschließend mit bereits vorhandenen Services zu einem Gesamtsystem zusammengefügt. In der Installationsphase (engl.: deployment) wird das Gesamtsystem installiert und ausgeführt. Während der Ausführung werden Laufzeitdaten aufgezeichnet, die wiederum als Grundlage für eine zukünftige Verbesserung des Modells gelten.

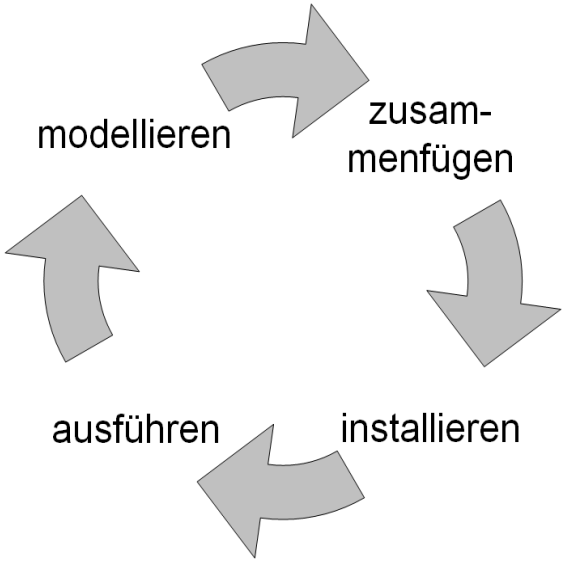

Abbildung 2: SOA-Lifecycle [IJC07]

Der Architekturstil SOA ist unabhängig von der verwendeten Technologie. Das Konzept der Web Services [W3C1] ist die wohl erfolgreichste Realisierung einer SOA, das im nächsten Abschnitt beschrieben wird.

2.1.3. Web Services

Das World Wide Web Consortium (W3C), das für zahlreiche Web Service Standards zuständig ist, definiert Web Services wie folgt:

> *"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 XML serialization in conjunction with other Web-related standards." [W3C2]*

Ein Web Service ist demnach eine Software, deren Operationen auf definierte Weise über eine Schnittstelle aufgerufen werden und benutzt die Technologien SOAP, HTTP, XML und WSDL. SOAP [W3C4] ist der Name für das plattformunabhängige Nachrichtenformat, das für die Strukturbeschreibung einer Nachricht und für den Aufruf eines Web Services verwendet wird. SOAP-Nachrichten werden über ein Transportprotokoll übertragen. Häufig kommt dabei das im Internet verwendete Hypertext Transfer Protocol (HTTP) [W3C5] oder deren sichere Variante (HTTPS) zum Einsatz. Mit der menschen- und maschinenlesbaren Extensible Markup Language (XML) [W3C3] können beliebige Inhalte strukturiert beschrieben werden.

Wie viele andere Web Service Standards baut die Web Service Description Language (WSDL) Version 1.1 [W3C6] auf XML auf und wird benutzt, um die Schnittstelle und die Implementierung eines Web Services zu beschreiben. Ein WSDL-Dokument besteht aus zwei Teilen. Der wieder verwendbare, plattformunabhängige Teil beschreibt, welche Operationen (`<portType>`) ein Web Service anbietet und welche Datentypen dabei verwendet werden. Nachrichten für eine Operation können in eine Richtung (engl.: one way) oder in beide Richtungen (engl.: request/ response) geschickt werden. Der konkrete Teil hingegen beschreibt wie und wo der Web Service aufgerufen werden kann. Dazu gehört die Festsetzung eines bestimmten Protokolls (`<binding>`), beispielsweise SOAP/ HTTP.

Um einen Geschäftsprozess zu beschreiben, reicht die Beschreibung der einzelnen Services jedoch nicht aus. Um ein bestimmtes Geschäftsziel zu erreichen, müssen die Aktivitäten in einer bestimmten Reihenfolge abgearbeitet werden. Wie eine solche Reihenfolge modelliert wird, erläutert der folgende Abschnitt.

2.2. Workflow Management und BPEL

2.2.1. Business Process Management

Der Begriff Business Process Management (BPM, auch: Business Process Modeling) [HAV05] beschreibt die Modellierung und den Umgang mit Geschäftsprozessen. Dazu gehören die Prozessdefinition, d.h. der Ablauf des Prozesses und die Definition der Aktivitäten. Aktivitäten können automatisiert oder manuell ausgeführt werden. Manuell ausgeführte Aktivitäten heißen im Fokus dieser Arbeit „Tasks“. Letztere werden von Benutzern ausgeführt, die aus Prozesssicht jeweils eine bestimmte Rolle einnehmen.

Die formale Modellierung eines Prozesses ermöglicht es, diesen besser zu verstehen und zu dokumentieren. Ist der Ablauf eines Geschäftsprozesses geeignet dokumentiert, erleichtert dies beispielsweise die Einarbeitung neuer Mitarbeiter. Besonders leicht verständlich ist die visuelle Darstellung eines Prozesses, die einem allgemein anerkannten Standard folgt. Ein solcher Standard ist die Business Process Modeling Notation (BPMN) [OMG1], die an die aus der Softwaremodellierung bekannten Sprache Unified Modeling Language (UML) [OMG2] angelehnt ist.

Die Geschäftsprozess-Modellierung ist außerdem Grundvoraussetzung für das Erkennen von Verbesserungspotentialen. Erhebliches Potenzial besteht oft darin, einzelne Prozessteile zu automatisieren oder auszulagern. So zeigen Fallstudien [FIS01], dass beispielsweise ein Finanzdienstleister seine Bearbeitungszeiten deutlich senken konnte und trotzdem ein Drittel seines Personals einsparte.

2.2.2. Workflow Management

Im Kontrast zu einem „Geschäftsprozess“, der meist von Geschäftsanalysten definiert wird, beschreibt ein „Workflow” [LR00] das auf einem Computersystem ausführbare Abbild eines Geschäftsprozesses. Auch wenn beide Begriffe oft synonym verwendet werden, wird diese Unterscheidung in dieser Arbeit verwendet. Workflow Management beschreibt die Erstellung und die Verwaltung von Workflows.

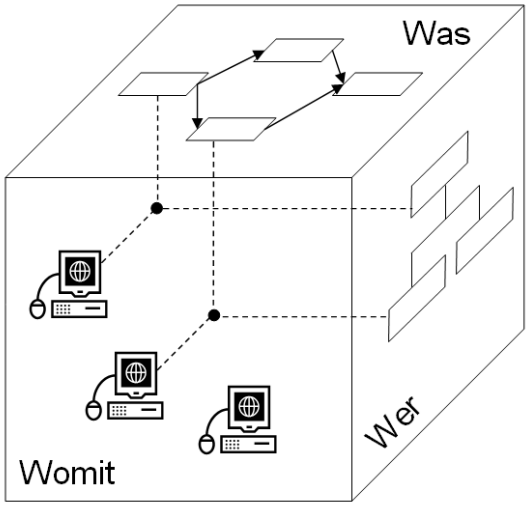

Abbildung 3: Workflow Dimensionen [LR00]

Ein Workflow besteht aus drei Dimensionen, die in Abbildung 3 dargestellt sind. Die erste Dimension (Was) beschreibt die Logik des Workflows, d.h. die Regeln, in welcher Reihenfolge die Aktivitäten abgearbeitet werden (Kontrollfluss) und die Daten, die von den einzelnen Aktivitäten benötigt werden (Datenfluss). Die zweite Dimension (Wer) beschreibt, die Organisation, in der der Workflow ablaufen soll, d.h. die Benutzer, die anhand der ihnen zugewiesen Rolle bestimmte Tasks abarbeiten. Die dritte und letzte Dimension (Womit) beschreibt die benötigten IT-Systeme; im Falle einer SOA die benötigten Web Services.

Ähnlich zum Lebenszyklus eines SOA Softwaresystems aus Abschnitt 2.1.2 kann ein Lebenszyklus eines Workflows beschrieben werden. Ein Geschäftsprozess wird grafisch in BPMN oder einer ähnlichen Sprache modelliert. Dieser Geschäftsprozess wird dann von einem technischen Analysten in ein ausführbares Workflow Modell umgewandelt [SCH08]. Dazu ist eine Verfeinerung (engl.: refinement) des Modells notwendig, beispielsweise die Adresse der aufzurufenden Web Services oder die Art der Fehlerbehandlung. Anschließend wird das Workflow Modell auf einer geeigneten Laufzeitumgebung („Workflow Engine“) installiert (engl.: deployed). Wird es nun aufgerufen, erstellt die Laufzeitumgebung eine neue „Instanz“ des Modells, d.h. ein Workflow, und führt diese aus. Die Workflow Engine überwacht und analysiert die ablaufenden Workflows und ermöglicht damit eine spätere Verbesserung des Geschäftsprozesses. Diesem Lebenszyklus folgt die Nutzung einer Workflow-zentrierten Applikation, die in Abbildung 4 dargestellt ist.

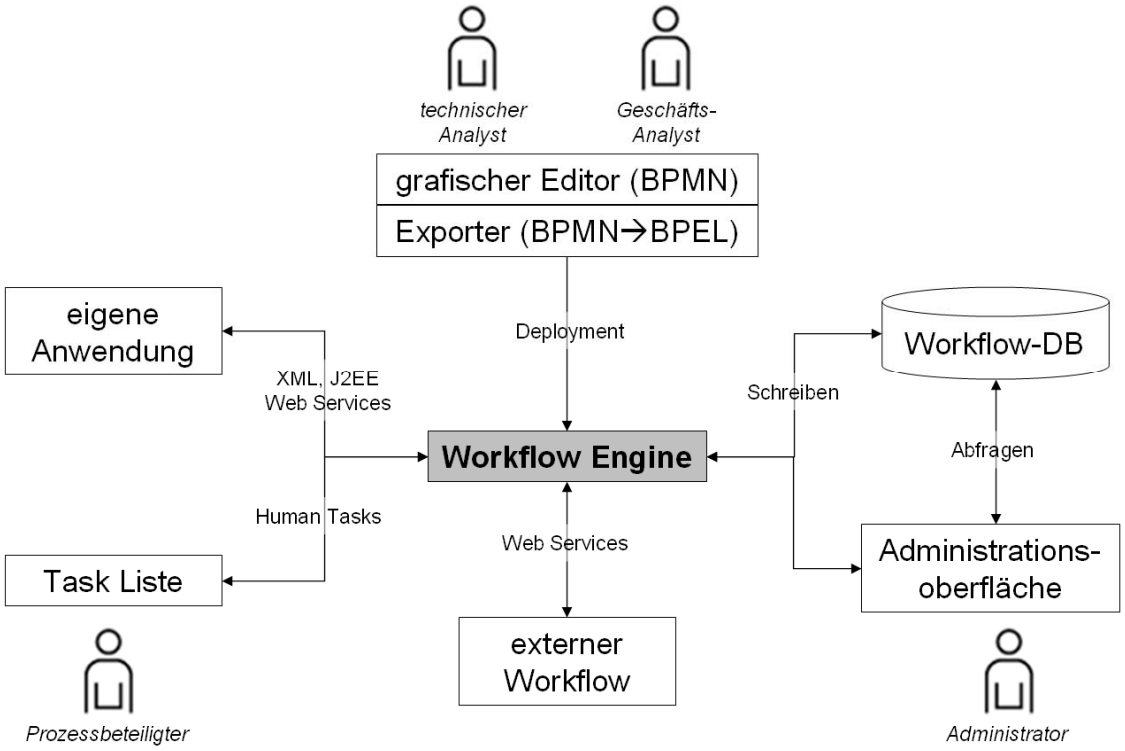

Abbildung 4: Workflow-zentrierte Applikation [HAV05]

Um einen Workflow auf einer Workflow Engine ausführen zu können, wird eine Sprache benötigt, um diesen zu beschreiben. Eine solche Sprache ist im folgenden Abschnitt beschrieben.

2.2.3. Business Process Execution Language

Die Web Services Business Process Execution Language (WS-BPEL, kurz: BPEL) [OAS2] ist eine von OASIS standardisierte XML-basierte Sprache zur „Orchestrierung“ von Web Services. Orchestrierung bezeichnet dabei das Vorgehen, einzelne Web Services so zu verbinden, dass ein Workflow entsteht. BPEL trennt damit das „Programmieren im Großen“, d.h. das Zusammenspiel einzelner Web Services, vom „Programmieren im Kleinen“, was in diesem Zusammenhang die Entwicklung einzelner Web Services bezeichnet. So entsteht in einem Architekturdiagramm (vgl. Abbildung 6) eine weitere Ebene, die so genannte „Workflow Ebene“. Diese enthält die Workflows, die die Web Services der darunterliegenden Service Ebene aufrufen. Web Services werden innerhalb des Workflows mithilfe von „Service Aktivitäten“ aufgerufen. „Strukturierte Aktivitäten“ gruppieren Aktivitäten und liefern die dafür notwendige Logik.

Zahlreiche freie und kommerzielle BPEL-Laufzeitumgebungen [HFS06] sind verfügbar und tragen zur wachsenden Bedeutung von BPEL und Web Services bei [GB06]. Ein BPEL `<process>`-Element beinhaltet laut den Aussagen aus Abschnitt 2.2.2 einen Workflow. In Tabelle 1 sind die Hauptbestandteile eines BPEL Workflows beschrieben. So genannte Handler behandeln das Verhalten im Fehlerfall, beispielsweise die Kompensierung eines Prozesses und verarbeiten Ereignisse.

Definierende Elemente	
`<partnerLinks>`	Definieren andere Web Services als Partner für eine gemeinsamen Interaktion
`<variables>`	Speichern Zustandsinformationen und dienen zur Weiterreichung von Daten innerhalb eines Prozesses
`<correlationSets>`	Identifizieren Daten in einem Prozess eindeutig, um zukünftige Nachrichten zuordnen zu können
Service Aktivitäten	
`<invoke>`	Ruft einen Web Service auf
`<receive>`	Startet eine Prozessinstanz, wenn ein Client Request empfangen wurde
`<reply>`	Sendet eine Antwort zum erhaltenen Client Request
`<assign>`	Verändert `<variables>`-Elemente
Strukturierte Aktivitäten	
`<scope>`	Dient als Container für andere Aktivitäten
`<pick>`	Startet eine Prozessinstanz oder setzt diese fort, wenn einer von mehreren möglichen Requests empfangen wurde
`<flow>`	Führt Aktivitäten entlang ihres Kontrollflusses oder parallel aus
`<sequence>`	Führt Aktivitäten in einer geordneten Reihenfolge aus
`<switch>`	Repräsentiert eine Entscheidung mit mehreren Ausgängen
`<while>`	Repräsentiert eine Schleife, die die beinhalteten Aktivitäten ausführt, solange eine bestimmte Bedingung erfüllt ist.

Tabelle 1: BPEL Hauptbestandteile

Die Erweiterbarkeit von BPEL durch zahlreiche Erweiterungen gegeben. So nutzt beispielsweise IBM in seinen Produkten die Erweiterungen BPELJ [IBM2] und BPEL4People [IBM3]. Mit BPELJ können Java Anweisungen und Objekte direkt aus dem Prozess aufgerufen werden, während BPEL4People mit WS-HumanTask die Einbeziehung von menschlichen Akteuren in einen Workflow ermöglicht.

Da ein Workflow im Sinne von BPEL ein Web Service ist, können Web Services und Workflows zusammengefasst werden. Dies beschreibt der nächste Abschnitt.

2.2.4. Service Component Architecture

Workflows, als auch Web Services können, wie in Abbildung 5 gezeigt, zu einer Komponente einer so genannten Service Component Architecture (SCA) [OSO1] abstrahiert werden.

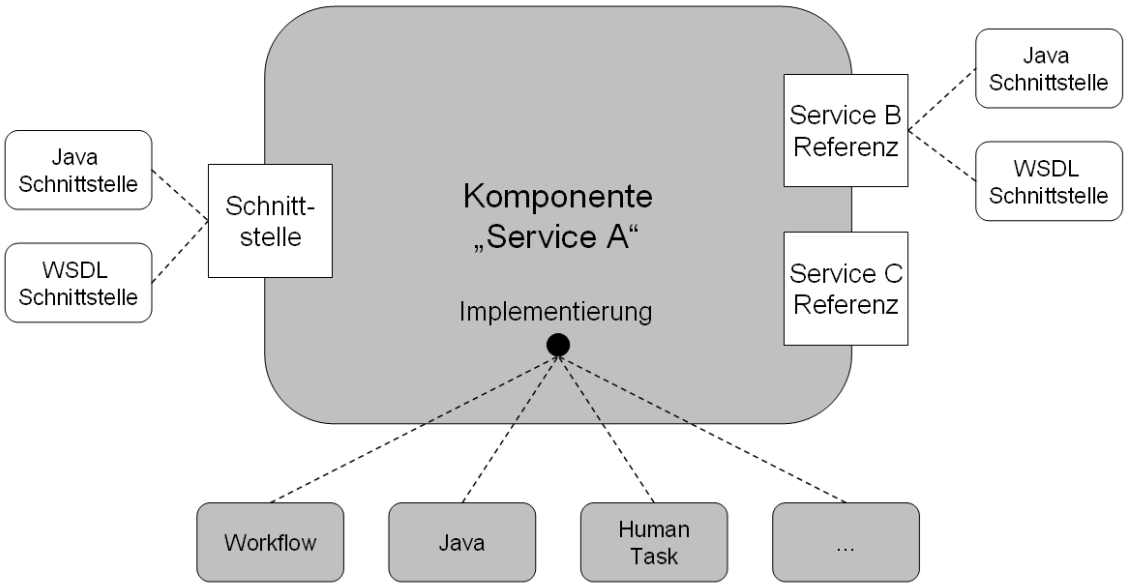

Abbildung 5: SCA-Komponente [IJC07]

Diese Komponenten haben eine Schnittstelle und Referenzen auf andere Komponenten. Da die interne Implementierung gekapselt bleibt, können die Komponenten miteinander verbunden und einzeln ausgetauscht werden. Dies kann die Integration mit bereits vorhandenen Systemen erheblich erleichtern. Die weiterzureichenden Daten werden passend zu den WSDL Schnittstellen in Service Data Objects (SDOs) [OSO2] definiert. Passen zwei Schnittstellen nicht zusammen, müssen die einzelnen Datentypen aufeinander abgebildet und gegebenenfalls transformiert werden. Dieses Vorgehen heißt Data-Mapping.

Der nächste Abschnitt beschreibt Software as a Service (SaaS). Dabei werden ganze Anwendungen zur Benutzung als Service angeboten.

2.3. SaaS und Variabilitäten

2.3.1. Software as a Service

Chong u.a. [CC06] definieren SaaS als Software, die als Service bereitgestellt wird und auf die über das Internet zugegriffen wird. Die Idee dabei ist, dass viele Kunden in diesem Zusammenhang auch Mandanten genannt, dieselbe Instanz einer Software benutzen. Die so genannte „Single-Instance Multi-Tenancy Architektur“ einer solchen Software ermöglicht eine effizientere Ressourcennutzung der bereitgestellten Anwendung. Kunden eines SaaS Anbieters müssen damit Software nicht mehr kaufen und auf eigenen Servern bereitstellen, sondern mieten die Software und bezahlen nur die tatsächliche Benutzung. Durch die Nutzung von Skaleneffekten können so Teile der Infrastruktur- und Softwarewartungskosten eingespart werden.

SaaS Anwendungen müssen skalierbar, mandantenfähig und konfigurierbar sein. Dies hat einen signifikanten Komplexitätsanstieg bei der Anwendungsentwicklung zur Folge. Außerdem ist ein Kunde, der eine solche Anwendung nutzt, abhängig vom jeweiligen Anbieter und gibt einen großen Teil seiner Verantwortlichkeiten und IT-Kompetenzen an den Anbieter ab. SaaS wird oft in Verbindung mit „Cloud Computing“ genannt, das in der Industrie mittlerweile breite Unterstützung [HEI1] gefunden hat.

Viele Kunden sind an Nischenprodukten oder an speziellen Dienstleistungen interessiert, die selten nachgefragt werden. Dieser so genannte „Long Tail“ [AND08] bietet sich besonders zur Auslagerung an einen SaaS Anbieter an. Die ausgelagerten Anwendungen müssen erweiterbar und skalierbar sein, um eine möglichst große Zahl an Kunden beliefern zu können. Frühere Ansätze, wie z.B. der des „Application Service Providers“ (ASP) konnten dies nicht in dem Maße leisten [SC02].

Der Reifegrad der Architektur einer SaaS Anwendung lässt sich laut [CC06] in vier Stufen aufteilen. Die erste Stufe entspricht dem ASP Modell, bei der jeder Mandant seine eigene Softwarevariante benutzt („Single Tenancy“). Dies hat den Vorteil, dass die Anwendung exakt nach den Wünschen des Mandanten erstellt werden kann, aber auch den Nachteil, dass der Mandant für die Software bezahlen muss, selbst wenn er sie nicht benutzt. In der zweiten Stufe wird die gleiche konfigurierbare Codebasis für alle Mandanten benutzt, was die Wartungskosten deutlich verringert. Im Gegensatz zu Stufe Zwei nutzen Mandanten in Stufe Drei eine einzige Instanz der Anwendung mit einer isolierten Datenbasis. Die höchste Stufe ist eine skalierbare SaaS Anwendung, bei der auf einer lastbalancierten Serverfarm identische Instanzen mit gemeinsamer Datenbasis und ohne Zustandsinformationen (engl.: stateless) laufen. Da eine gemeinsame Datenbasis aus Datenschutzgründen nicht immer möglich ist, muss ein entsprechender Kompromiss zwischen Datenisolation und Skalierbarkeit gefunden werden. Abbildung 6 gibt einen groben Überblick für eine typische SaaS Architektur.

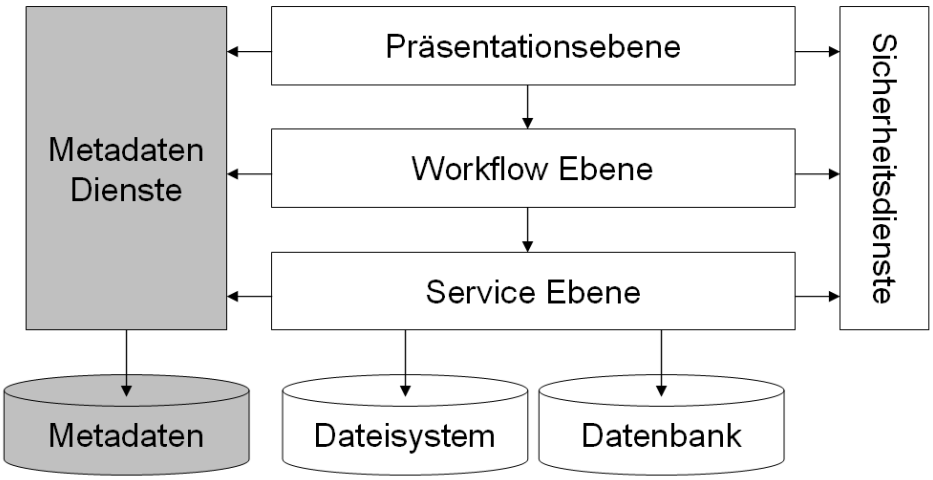

Abbildung 6: SaaS-Architektur [CC06]

Der wichtigste Unterschied zwischen einer herkömmlichen und einer SaaS Architektur ist die Verwendung von „Metadaten“, die zur Konfiguration der einzelnen Anwendungsinstanzen dienen. Jeder Mandant kann seine Instanz konfigurieren. Dazu gehört eine anpassbare Oberfläche, so dass er seine Anwendung an die Vorgaben seiner Firma anpassen kann, sowie anpassbare Geschäftsregeln und Abläufe. Auch das Datenmodell muss erweiterbar sein, beispielsweise durch vom Mandanten konfigurierbare Tabellen oder Tabellenspalten. Die Authentifizierung ist entweder zentral oder dezentral umsetzbar. Eine zentrale Authentifizierung ist einfacher zu implementieren, eine dezentrale Lösung hingegen ist benutzerfreundlicher, da der Kunde im günstigsten Fall nur einmal seine Zugangsdaten eingeben muss („Single Sign-On“).

Skalierbarkeit ist eine zentrale Anforderung an eine SaaS Anwendung. Dies hat zur Folge, dass für eine einfache Verteilung auf Zustandsinformationen möglichst verzichtet wird. Serviceaufrufe sind häufig asynchron, um eine gemeinsame Ressourcennutzung zu ermöglichen. Idealerweise greifen alle Mandanten auf die gleiche, konfigurierbare und verteilte Anwendungsinstanz zu. Ein weiterer wichtiger Erfolgsfaktor ist eine gute operative Struktur. Diese wird durch gemeinsam genutzte Services ermöglichst, die die Verfügbarkeit und die aktuelle Performance überwachen und wenn nötig, entsprechende Maßnahmen ergreifen. Auf diese Weise kann auch die Einhaltung von Service Level Agreements (SLAs) überprüft werden. Verfügbarkeit ist das wohl wichtigste SLA, da ein Mandant durch mit einer SaaS Anwendung nicht offline arbeiten kann.

2.3.2. eCCo als SaaS Anwendung

Der Prototyp eCCo, der in dieser Arbeit verwendet wird, kann als SaaS Anwendung angesehen werden. Der SaaS Anbieter ist in diesem Fall der Autohersteller, der seine Produkte vertreiben möchte. Typischerweise gehören zu einem international tätigen Autohersteller eine oder mehrere Vertriebsgesellschaften pro Markt. Diese stellen die Mandanten innerhalb des Szenarios dar. Ein Autohändler ist der Kunde einer Vertriebsgesellschaft und Benutzer der SaaS Anwendung.

Jeder Markt hat eigene Gesetze, Fahrzeugkonfigurationen und Preisberechnungsschemata. Die unterschiedlichen Mandanten haben demnach verschiedene Anforderungen und müssen ihre Anwendungsinstanz konfigurieren können. Eine Bestellanwendung muss in regelmäßigen Abständen weiterentwickelt werden, um auf geänderte Marktsituationen reagieren oder neue Rechtsrahmenbedingungen erfüllen zu können. Werden beispielsweise vom Rat der Europäischen Union neue Gesetze erlassen, so gelten diese für alle Märkte innerhalb Europas. Wenn jeder Markt seine eigene Bestellsoftware nutzt, dann muss im Falle einer Änderung der Bestellabläufe jedes dieser Systeme geändert werden. Da Bestellsysteme meist mit anderen Systemen, wie Rechnungs-, Buchhaltungs- und Garantieabwicklungssystemen verbunden sind, steigt der Wartungs- und Integrationsaufwand erheblich. Es kann sich durchaus lohnen, dass ein Hersteller seine Vertriebssoftware zentral verwaltet und weiterentwickelt.

Eine Anpassbarkeit der Oberfläche durch den Kunden sowie die Benutzung derselben Anwendung über mehrere Hersteller hinweg liegt nicht im Fokus dieser Arbeit. Letzteres wird auch schwer erreichbar sein, da die meisten Hersteller wohl sicher nicht ihre Vertriebsanwendungen mit der Konkurrenz teilen möchten.

Um von Skaleneffekten profitieren zu können, muss dieselbe Anwendungsinstanz von verschiedenen Mandanten benutzt werden können. Dabei können Multi-Tenancy-Patterns helfen, die im folgenden Abschnitt beschrieben sind.

2.3.3. Multi-Tenancy-Patterns

Patterns [FDF02, HW03] sind in der Softwarewelt weit verbreitet und dienen dem besseren Verständnis von komplexen Softwarestrukturen. Sie vereinfachen damit die Einarbeitung und die Wartung einer Anwendung und steigern damit die Qualitätseigenschaften dieser Anwendung. Wie in Abschnitt 2.3.1 beschrieben, ist eine zentrale Eigenschaft einer SaaS Anwendung die Mandantenfähigkeit. [MUT09] beschreibt Patterns für Web Services, die dies ermöglichen (Multi-Tenancy-Patterns).

Da verschiedene Mandanten unterschiedliche Anforderungen an eine SaaS Anwendung haben, muss eine solche flexibel anpassbar sein. Anpassbarkeit zieht sich durch alle Ebenen einer SaaS Architektur. Die Präsentationsschicht muss anpassbar sein, genauso wie die Web Services der Workflow- und Serviceebene. Letztlich müssen auch die Datenbank und die Schnittstellen zu Fremdsystemen Möglichkeiten zur Anpassung bieten. Die in [MUT09] beschriebenen Multi-Tenancy-Patterns sind auf die Serviceebene in einer SOA-basierten SaaS Anwendung anwendbar. Das heißt, sie beschreiben, wie mandantenfähige Web Services aufgebaut (Service Patterns) und aufgerufen (Invocation Patterns) werden können. Die verschiedenen Service Patterns werden in Abbildung 7 dargestellt.

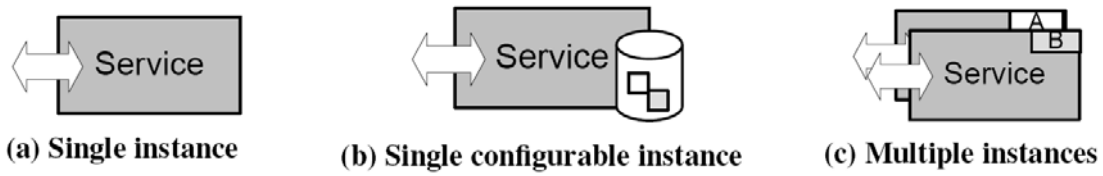

Abbildung 7: Service Patterns [MUT09]

Ein Single Instance Service verhält sich für alle Mandanten gleich, d.h. er ist nicht anpassbar. Derartige Web Services sind besonders gut skalierbar, da identische Instanzen auf mehrere Server verteilt werden können. Auch sind sie sehr gut wartbar, da bei einer Änderung nur ein Web Service neu installiert werden muss. Mandantenspezifisches Verhalten oder eine Isolation der Daten ist mit Single Instance Services jedoch nicht möglich. In diesem Fall bietet sich ein Single Configurable Instance Service an. Mandantenspezifisches Verhalten wird dabei über Konfigurationsdaten erreicht. Wird ein neuer Mandant hinzugefügt, so muss dieser Service für den neuen Mandanten verfügbar sein, ohne dass er neu kompiliert oder installiert werden muss. Die Konfigurationsdaten (auch: Mandantenkontext) werden gemeinsam mit dem Web Service installiert oder in einer gemeinsamen Datenbank abgelegt. Die Implementierung muss dann anhand der Konfigurationsdaten das Verhalten zur Verfügung stellen, das zum aktuellen Mandaten gehört. Im Falle einer Änderung müssen nur die Konfigurationsdaten, nicht aber die Implementierungen angepasst werden, was die Wartbarkeit erleichtert. Auch die Skalierbarkeit ist gegeben, da mehrere Mandanten auf dieselbe Instanz eines Services zugreifen können. Wartbarkeit und Skalierbarkeit werden allerdings durch die Nutzung Konfigurationsdaten eingeschränkt. Eine gemeinsam genutzte Datenbank kann Performanceeinbußen verursachen und verteilte Konfigurationsdaten müssen bei Änderungen bei allen Instanzen geändert werden.

Sollte das durch einen Web Service implementierte Verhalten pro Mandant so unterschiedlich sein, dass ein Single Configurable Instance Service nicht mehr sinnvoll benutzbar ist, werden Multiple Instances Services benötigt. Ein weiterer Grund kann sein, dass Web Services private Daten benutzen, auf die nur von einem einzelnen Mandanten zugegriffen werden darf. Jeder Mandant oder jede Gruppe von Mandanten benutzt dabei eine eigene Implementierung eines Web Services. Multiple Instances Services korrumpieren die Idee einer Single-Instance Multi-Tenancy Architektur, denn solche Services können nicht einfach für die lastbalancierte Benutzung verteilt werden. Auch müssen, wie beim ASP Modell bei einer Änderung alle Instanzen dieser Services aktualisiert werden. Dieser Umstand kann teilweise umgangen werden, wenn so genannte Templates verwendet werden, die die Teile enthalten, die für alle Instanzen gleich sind. Da BPEL-Laufzeitumgebungen mandantenspezifische Konfigurationsmöglichkeiten für Workflows oft nicht ermöglichen bzw. kein mandantenspezifisches Monitoring ermöglichen, ist auch für Workflows das Multiple Instances Service Pattern zu verwenden.

Da in einer SOA Wiederverwendung eines der wichtigsten Konzepte ist, können aufzurufende Web Services auch von außerhalb des eigenen Unternehmens bereitgestellt werden. Diese externen Web Services können in Verbindung mit den eigenen Web Services verwendet werden, wobei unter Umständen unterschiedliche Schnittstellen zum Einsatz kommen, die aufeinander abgebildet werden müssen. Für diese Abbildung gibt es Enterprise Application Integration (EAI) Patterns, wie beispielsweise Message Translator oder Content Enricher, die in [HW03] beschrieben werden. Abbildung 8 stellt die in [MUT09] eingeführten Invocation Patterns dar.

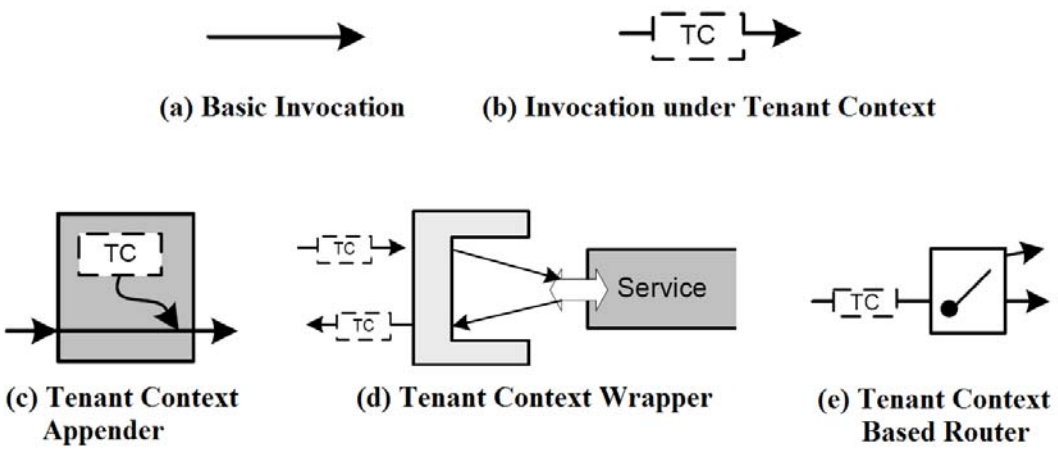

Abbildung 8: Invocation Patterns [MUT09]

Basic Invocation beschreibt den klassischen Aufruf eines Web Services ohne Mandantenkontext. Jeder nicht mandantenfähige Web Service wird so aufgerufen. Verhält sich der aufzurufende Web Service mandantenspezifisch, so wird das Invocation under Tenant Context Pattern benutzt. Dabei fügt der aufrufende Web Service den Mandantenkontext in die Aufrufnachricht ein, was voraussetzt, dass sowohl der aufrufende, als auch der aufgerufene Web Service den Mandantenkontext verarbeiten kann. Da die meisten Web Services ohnehin Autorisierungsmechanismen bereitstellen, können diese für den Mandantenkontext verwendet werden.

Wenn der aufrufende Web Service nicht verändert werden darf und dennoch Mandantenkontext für einen Aufruf benötigt wird, kann ein „Tenant Context Appender“ Service verwendet werden. Dieser wird anstelle des aufzurufenden Web Services aufgerufen, reichert seine Eingangsnachricht mit dem Mandantenkontext an und leitet sie an den Web Service weiter. Ein Tenant Context Appender kann auch ein Data-Mapping für den aufzurufenden Web Service durchführen. Diesen Spezialfall nennt man „Tenant Context Mediator“. Ein weiterer Spezialfall ist der „Tenant Context Wrapper“. Dieser kapselt den Mandantenkontext vollständig. So kann beispielsweise ein BPEL-Prozess als Single Instance Service installiert werden, während der Tenant Context Wrapper einen Mandantenkontext zu allen mandantenspezifischen Aufrufen hinzufügt, d.h. der BPEL Prozess kann Single Configurable Instance Services aufrufen, ohne dass das Prozessmodell geändert werden muss.

Zum Aufruf von Multiple Instances Services oder von verschiedenen externern Web Services wird der Tenant Context Based Router verwendet. Dieser leitet empfangene Nachrichten abhängig vom Mandantenkontext an den jeweiligen Web Service oder die jeweilige Instanz weiter. Der Tenant Context Based Router ist damit eine Spezialisierung des Content Based Router Patterns aus [HW03].

2.3.4. Variabilitäten

Variabilitäten sind ein Konzept aus dem Bereich der Software-Produktlinien [PM08]. Die Entwicklung von Software-Produktlinien erlaubt Kostensenkungen, Qualitätssteigerungen und verkürzte Entwicklungszeiten durch Wiederverwendung. Dabei wird die Softwareentwicklung in zwei Bereiche aufgeteilt. Ein Bereich beschreibt die Teile, die gemeinsam verwendet werden. Diese werden in dieser Arbeit „Templates“ genannt. Der andere Bereich, die „Variabilitäten“, beschreibt, „wie sich die verschiedenen Produkte der Produktlinie bezüglich ihrer Eigenschaften unterscheiden können“. Dieser Ansatz ist ähnlich zur Benutzung einer abstrakten Oberklasse in einer objektorientierten Programmiersprache. Um ein neues Produkt aus einer Produktlinie zu erzeugt, müssen alle Variabilitäten „gebunden“ werden, d.h. es muss festgelegt werden, wie die variablen Teile des Produkts sich verhalten. Die dadurch entstehende Instanz der Anwendung enthält nur noch Ausprägungen der einzelnen Variabilitäten und keine variablen Teile mehr.

Da Variabilitäten während des gesamten Entwicklungsprozesses berücksichtigt werden müssen, können Variabilitäten innerhalb von UML Anwendungsfall-, Klassen- und Aktivitätsdiagrammen [OMG2] modelliert werden. Um die verschiedenen Modellierungsarten konsistent zu halten, können die einzelnen Varianten zu so genannten „Variabilitätspunkten“ zusammengefasst werden. Dieses Vorgehen wird „Orthogonale Variabilitätsmodellierung“ genannt. Die Elemente eines solchen Modells werden in Abbildung 9 dargestellt. Variabilitäten werden dabei explizit, d.h. über den gesamten Entwicklungsprozess hinweg, anstatt implizit über mehrere Modelle verteilt modelliert. Variabilitäten können mit so genannten „Constraints“ eingeschränkt werden. Diese beschreiben beispielsweise, ob eine Variante obligatorisch oder optional an ein Produkt gebunden wird. Externe Variabilitäten sind solche, die für den Kunden sichtbar sind, während interne Variabilitäten aus technischen Gründen eingeführt werden. Es ist sinnvoll, die Begründung für die Einführung einer Variabilität zu dokumentieren.

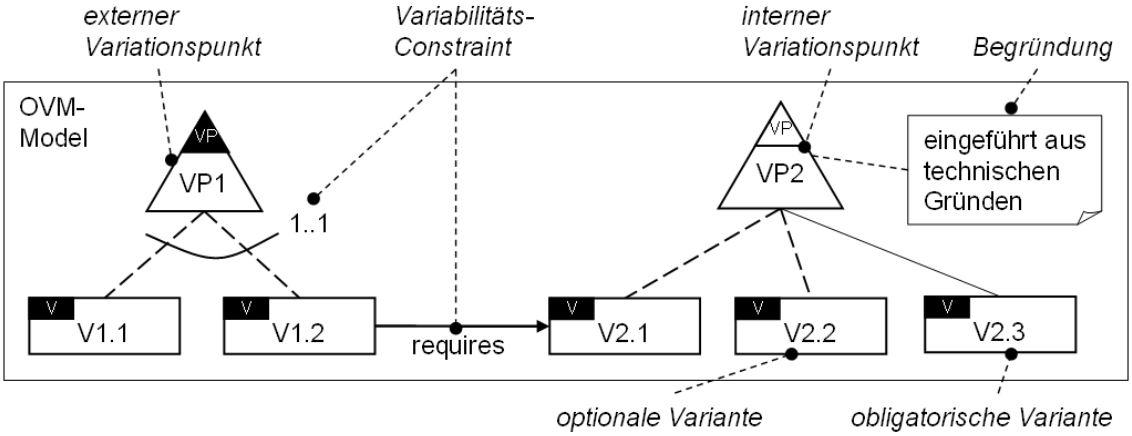

Abbildung 9: Orthogonale Variabilitätsmodellierung [PM08]

Mietzner u.a. [MML09] beschreiben, wie die explizite Modellierung von Variabilitäten für die Entwicklung von SaaS Anwendungen genutzt werden kann. Diese unterstützt SaaS Anbieter bei der Entscheidung, welche Varianten sie ihren bestehenden und zukünftigen Kunden anbieten werden. SaaS als Geschäftsmodell lohnt sich nur dann, wenn ein ausreichend breites Kundenspektrum angesprochen werden kann. Da Letzteres viele verschiedene Anforderungen zur Folge hat, muss ein Kompromiss zwischen einem Grad an Flexibilität und der Möglichkeit diese Flexibilität verwalten zu können gefunden werden. Jeder Mandant muss dabei einen „Anpassungsprozess“ durchlaufen, bei dem er zwischen den zur Verfügungen stehenden externen Variabilitäten wählen kann. Nach Abschluss des Anpassungsprozesses werden alle externen Variabilitäten gebunden und der Mandant kann die nach seinen Wünschen angepasste Anwendung nutzen. Die voneinander abhängigen Varianten werden durch vorher gepflegte Einschränkungen erkannt. Interne Variabilitäten werden automatisch anhand vorher definierter Kriterien gebunden, so dass die für den Anbieter möglichst günstigste Variante gewählt wird. Auf diese Weise kann eine automatische Installation von mandantenspezifischen Anwendungsinstanzen erreicht werden, d.h. ein Mandant kann seine Anwendung innerhalb definierter Grenzen selbst anpassen, ohne dass weitere Entwicklungsarbeiten nötig wer-

den. Wenn ein Mandant seine Anwendungsinstanz nicht mehr benötigt, kann diese automatisch wieder entfernt werden.

Wie in Abschnitt 2.3.3 dargelegt, muss eine SaaS Anwendung auf jeder Ebene der Architektur anpassbar sein. Eine Variabilität auf Präsentationsebene beschreibt beispielsweise die Auswahl der Sprache für die Benutzeroberfläche. Auf der Ebene der Workflows können Benutzer verschiedene Varianten eines Ablaufs wählen. So kann es beispielsweise bei einem Kunden nötig sein, dass an einer bestimmten Stelle in einem Workflow ein Prüfschritt notwendig ist, den ein anderer Kunde nicht durchführen möchte. Ist die SaaS Anwendung auf Basis einer BPEL Engine realisiert, kann ein so genannter abstrakter Workflow entworfen werden, der für alle Mandanten gleich ist. BPEL 2.0 [OAS2] spezifiziert sogenannte „opake Merkmale“ (engl.: opaque tokens) eines Workflows, die dem Konzept der Variabilitätspunkte sehr ähnlich sind [ML08a, ML08b]. Die opaken Merkmale müssen gebunden werden, bevor der Workflow ausgeführt werden kann. Leider erlauben sie nicht die Erstellung von Einschränkungen oder Abhängigkeiten, die nötig sind, um einen automatisierten Anpassungsprozess durch den Kunden zu ermöglichen. Beispielsweise muss oft die Benutzeroberfläche entsprechend angepasst werden, wenn ein Variabilitätspunkt auf Workflowebene gebunden wird.

In einem SaaS Szenario müssen Variabilitäten explizit und inklusive aller Abhängigkeiten modelliert werden. Mietzner u.a. [ML08a] definieren so genannte „Variabilitätsbeschreibungen“, aus denen der Anpassungsprozess für einen SaaS Mandaten erzeugt werden kann. Diese Variabilitätsbeschreibungen enthalten die Variabilitätspunkte und Verweise auf die entsprechenden Stellen im Template. Variabilitätspunkte können dabei voneinander abhängig sein. Zum Anpassungsprozess gehört auch eine „Validitätsprüfung“, d.h. eine Prüfung ob die ausgewählten Varianten miteinander kombinierbar sind.

Das Konzept der Erstellung von Produkten aus einer Produktlinie lässt sich mit der Forderung nach einer mandantenfähigen Anwendung nur schwer vereinbaren, weil für jeden Mandanten eine eigene Instanz der Anwendung erstellt werden würde. Demnach würden alle Anwendungen dem Multiple-Instance Pattern folgen, was wie in Abschnitt 2.3.3 beschrieben, die Idee einer Single-Instance Multi-Tenancy Architektur korrumpiert. Die von einem Workflow aufgerufenen Services können ebenfalls variabel sein. Da innerhalb einer SCA [OSO1] Workflows und Services als gleichwertige Komponenten betrachtet werden, bietet sich diese für die Erstellung von konfigurierbaren Anwendungspaketen [MLP08] an. SCA enthält durch die Beschreibung der Schnittstelle bereits einen einfachen Mechanismus zur Konfiguration von Komponenten. Dieser kann so erweitert werden, dass er eine vollständige Variabilitätsbeschreibung für eine Komponente darstellt. Mithilfe der SCA Referenzen, die zur Laufzeit ausgewertet werden können, wird auf die variablen Teile verwiesen. Man erhält dadurch Komponenten, die zur Laufzeit wiederverwendet und damit besser skaliert werden können, mit anderen Worten: Single Configurable Instances.

Einen ähnlichen Ansatz verfolgt die Software PESOA [OSL06], ein Framework zur modell- und prozessgetriebenen Softwareentwicklung auf Basis einer SOA. In der studentischen Fallstudie werden Varianten eines Geschäftsprozesses grafisch in BPMN [OMG1] modelliert. Es entstehen dadurch so genannte „variantenreiche Prozessfamilien“. Aus diesen werden dann ein „variantenfreies Prozessmodell“ und die zugehörige Webapplikation erzeugt. Da grafische Modelle sowohl von Fachbereichen als auch von IT-Entwicklern verstanden werden, bilden sie ein sehr gutes Kommunikationsmittel, um Variabilitäten zu dokumentieren. In BPMN modellierte Geschäftsprozesse lassen sich automatisch BPEL Workflows umwandeln [SCH08]. Um Konsistenz zwischen Geschäftsprozessen und der Anwendung sicherzustellen, wird in dieser Arbeit eine ähnliche Vorgehensweise gewählt. Doch zunächst wird in Kapitel 3 die zu erstellende Anwendung spezifiziert.

3. Spezifikation

Das folgende Kapitel widmet sich der Spezifikation der im Rahmen dieser Arbeit entwickelten Anwendung. Die Anforderungen an die zu erstellende Anwendung werden in Abschnitt 3.1 beschrieben. Abschnitt 3.2 beschreibt in Form von Anwendungsfällen, wer die Anwendung benutzt und wie sie benutzt wird. Auf der Grundlage des Datenmodells aus Abschnitt 3.3 werden in den Abschnitten 3.4 und 3.5 die fachlichen Prozesse und ihre variablen Teile beschrieben.

3.1. Anforderungen

3.1.1. Ziel und Zweck des Prototyps

Um die in Abschnitt 2.3 erläuterten Konzepte zu verifizieren, wird ein Prototyp erstellt, dessen Anforderungen in diesem Kapitel spezifiziert werden. ECCo dient als Technologieprototyp für ein Autohändlersystem. Der Urheber dieser Anwendung, die GFI-iTmedia Consult GmbH, unterstützt seit geraumer Zeit Automobilhersteller bei der Anbindung von Autohändlersystemen [GFI1] und kann daher auf fundiertes Geschäftsprozesswissen in diesem Bereich zurückgreifen. Neue Technologien werden dafür zunächst im eigenen Haus evaluiert, bevor sie in der Praxis zum Einsatz kommen. Der im Rahmen dieser Arbeit erstellte Prototyp orientiert sich an vorherigen Versionen von eCCo. Dazu gehören die Abläufe, das Datenmodell und die Benutzeroberfläche. Aufgrund des begrenzten Zeitrahmens dieser Arbeit wird von einer aufwändigen Anpassung der verwendeten Werkzeuge abgesehen.

Der Einsatz der Anwendung in der Praxis ist nicht vorgesehen. Nichtfunktionale Anforderungen werden deshalb in dieser Arbeit nicht detailliert beschrieben. Auf Wart- und Erweiterbarkeit der Anwendung wird Wert gelegt. Dazu gehören die Verwendung von Coderichtlinien sowie die Verwendung einer Service-Orientierten Architektur, die die Modularität der Anwendung gewährleistet. Wie in einer produktiv eingesetzten Software wird ein hoher Grad an Skalierbarkeit angestrebt.

Das Ziel der Anwendung ist es, Angebots- und Bestellprozesse von Autohändlern abzubilden, die sich von Markt zu Markt unterscheiden können. Die Angebote und Bestellungen können nach der Erstellung verwaltet werden. Die Erstellung einer Benutzeroberfläche sowie die Pflege von Kunden und Fahrzeugen liegen nicht im Fokus dieser Arbeit. Zum Zweck die Ergebnisse besser präsentieren zu können, wird jedoch eine Benutzeroberfläche mitgeliefert. Um die Bezeichnungen konsistent mit der Implementierung zu halten, werden die Abbildungen für Akteure, Anwendungsfälle, Daten und Geschäftsprozesse in dieser Arbeit in englischer Sprache dargestellt. Der folgende Abschnitt beschreibt die fachlichen Anforderungen an den Prototyp.

3.1.2. Fachliches Sollkonzept

Ein Autohändler verkauft Fahrzeuge an Kunden, die von einem Fahrzeughersteller produziert werden. Zum Zweck der Information für den Kunden wird vom Autohändler ein Angebot erstellt. Nimmt dieser das Angebot an, wird aus diesem eine Bestellung. Im Fokus dieser Arbeit können Angebote und Bestellungen nur mit Bezug auf „Bestandsfahrzeuge“ erstellt werden. Letztere sind im Bestand einer Landesniederlassung oder des Händlers selbst und besitzen eine Fahrgestellnummer, die das Fahrzeug eindeutig identifiziert.

Ein Angebot (engl.: offer) kann bestätigt oder zur Bestellung (engl.: order) konvertiert werden. Sowohl ein Angebot, als auch eine Bestellung werden durch die Unterschrift des Kunden bestätigt (engl.: confirmed). Die Bestätigung des Angebots ist optional. Anschließend wird die Bestellung versendet und vom Disponenten akzeptiert oder abgelehnt. Wird sie abgelehnt, so wird sie wieder in ein Angebot zurückkonvertiert. Nach Auslieferung des Fahrzeugs an den Kunden wird die Bestellung zum Vertrag (engl.: contract). Angebote, Bestellungen und Verträge werden als Geschäftsdokumente angesehen, denen ein Geschäftszustand (engl.: business state) und ein Bewilligungszustand (engl.: approval state) zugeordnet wird. Mögliche Bewilligungszustände sind initiiert (engl.: init), bestätigt (engl.: confirmed), akzeptiert (engl.: accepted) und abgelehnt (engl.: rejected). Abbildung 10 stellt diese alle Geschäfts- und Bewilligungszustände und deren Übergänge als Zustandsgraph dar.

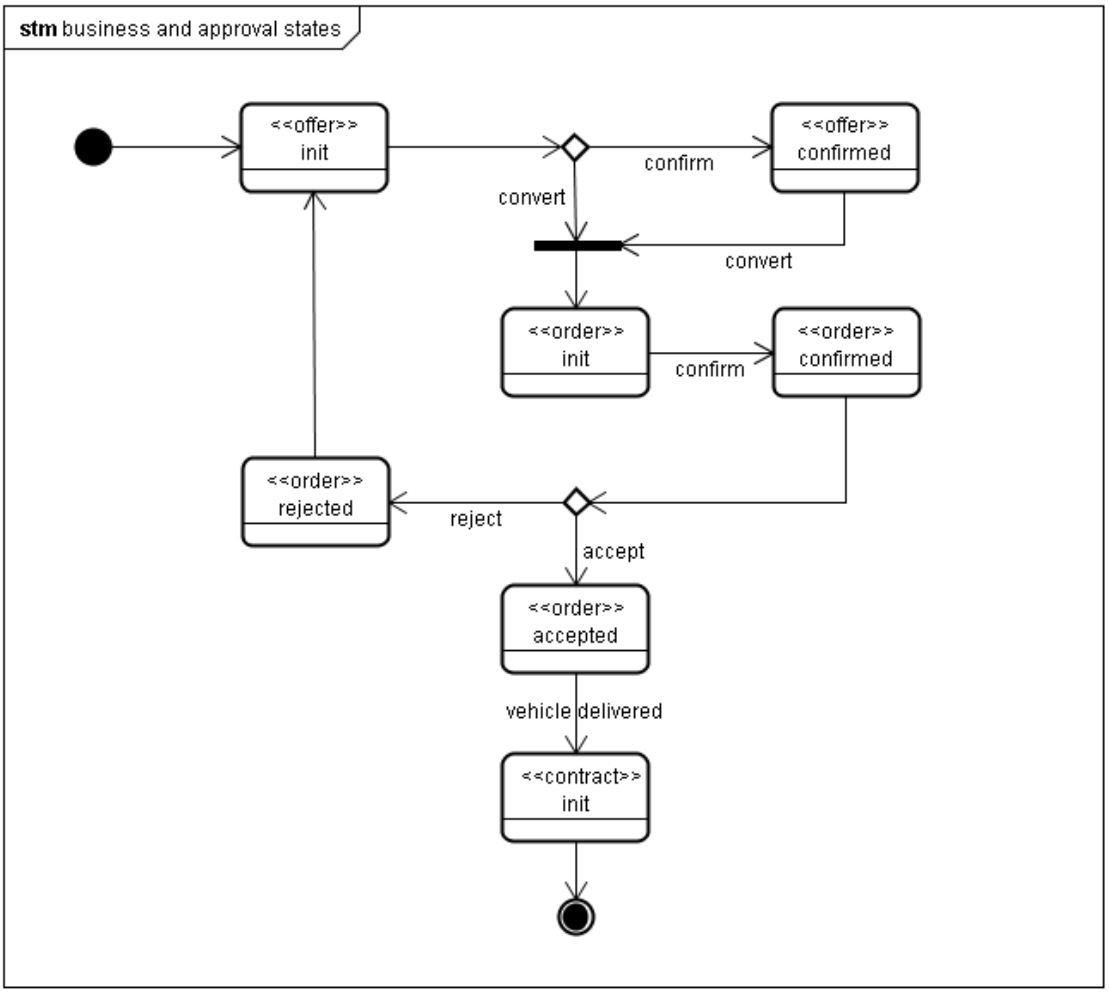

Abbildung 10: Geschäfts- und Bewilligungszustände

Teile eines Geschäftsdokuments sind nicht nur Kunden, Händler und Fahrzeuge, sondern häufig auch Dienstleistungen, die zwischen Kunde und Händler bzw. zwischen Kunde und Dritten vereinbart werden. Werden sie von einem Händler angeboten, so kann dieser Händler den Preis bestimmen. Dienstleistungen können nur auf bestimmte Weisen miteinander kombiniert werden. Während der Preis je nach Händler variieren kann, bestimmt der Fahrzeugstatus, d.h. ob das Fahrzeug neu oder gebraucht ist, die Menge der angebotenen Dienstleistungen. Letztere werden zu Dienstleistungsklassen zusammengefasst. Tabelle 2 beschreibt die Dienstleistungsklassen Finanzierung (engl.: financing), Leasing und Dienstleistung nach Vertragsabschluss (engl.: aftersales service).

Financing	Der Kunde finanziert das Fahrzeug mit einem Kredit.	
	Financing Standard	Finanzierung ohne Schlussrate
	Financing Balloon	Finanzierung mit Schlussrate
Leasing	Das Fahrzeug wird dem Kunden gegen ein Leasingentgelt zur Nutzung überlassen.	
	Leasing Standard	Leasing ohne Schlussrate
	Leasing Balloon	Leasing mit Schlussrate
Aftersales Service	Dienstleistungen betreffen den Zeitraum nach der Auslieferung des Fahrzeugs an den Kunden.	
	maintenance	Wartung
	repair and maintenance	Reparatur und Wartung
	tire replacement	Reifenersatz
	vehicle replacement	Ersatzfahrzeug
	fuel card	Tankkarte
	warranty extension	Garantieverlängerung

Tabelle 2: Dienstleistungsklassen

Die Kombinationsmöglichkeiten von Dienstleistungen sind eingeschränkt, d.h. hat der Kunde sein Fahrzeug mit einem Kredit finanziert, kann er keine Aftersales Services wählen. Bei Gebrauchtfahrzeugen ist keine Garantieverlängerung möglich.

Der Preis einer Dienstleistung ist ein Betrag, der sich ändern kann, wenn die Dienstleistung von einem bestimmten Händler angeboten wird. Der Fahrzeugpreis wird durch die Summe über den Basispreis eines Fahrzeugs und der Preise für die Sonderausstattungen berechnet. Der Gesamtpreis einer Bestellung ist die Summe über den Fahrzeugpreis und die Preise für die Dienstleistungen addiert mit einem prozentualen Mehrwertsteuersatz, der abhängig vom Land des Händlers ist.

Nach der Darstellung des fachlichen Konzepts beschreibt der folgende Abschnitt die Akteure, die mit der Anwendung interagieren, sowie die zugehörigen Anwendungsfälle.

3.2. Darstellung der Anwendungsfälle

3.2.1. Akteure

Die für die Anwendung relevanten Akteure sind Kunden, Händler und Disponenten. Der Kunde (engl.: customer) möchte ein Auto kaufen oder sich ein Angebot erstellen lassen. Der Händler (engl.: dealer) muss dafür die Kundendaten aufnehmen und den Geschäftsprozessen für die Angebotserstellung und Bestellung folgen. Bevor eine Bestellung abgeschlossen wird, muss sie überprüft werden. Dafür überprüft ein Disponent (engl.: sales admin) alle Bestellungen eines Händlers.

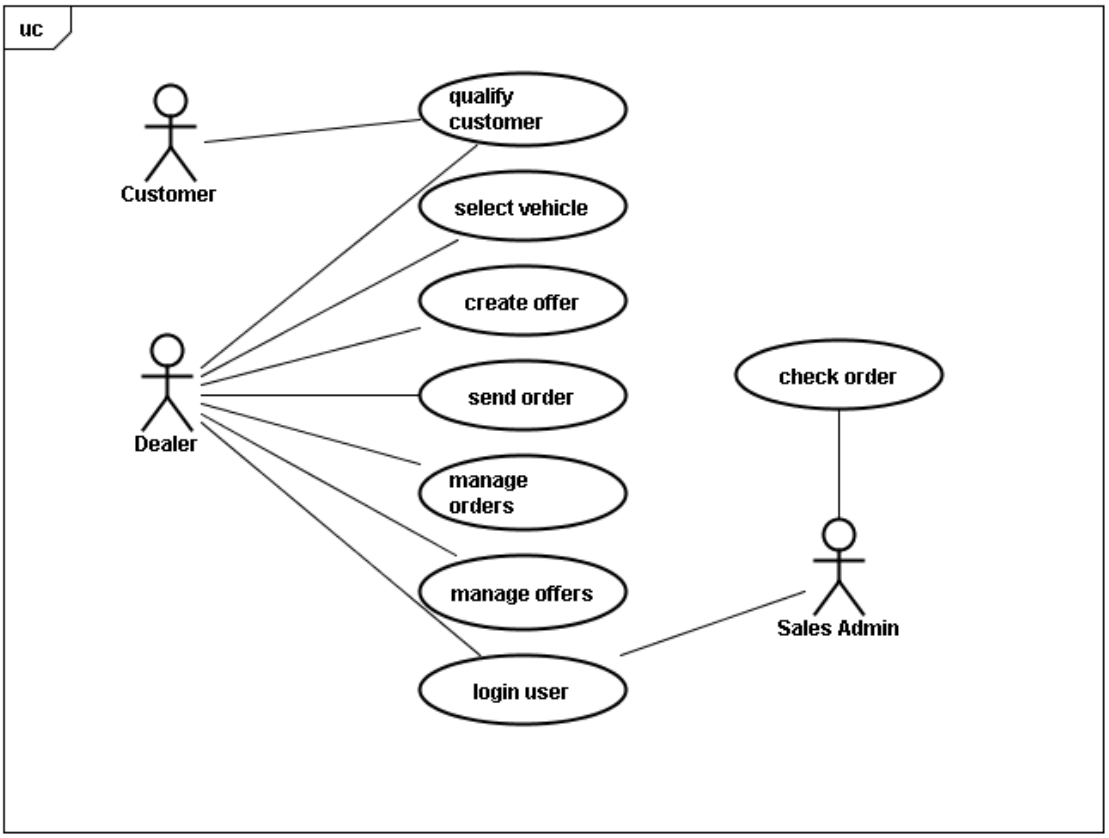

Abbildung 11: Anwendungsfälle

Abbildung 11 zeigt die beschriebenen Akteure in Form eines UML Anwendungsfalldiagramms. Die ihnen jeweils zugeordneten Anwendungsfälle werden im folgenden Abschnitt beschrieben.

3.2.2. Anwendungsfälle

Ein Anwendungsfall [COC00] beschreibt ein Abkommen zwischen einem Benutzer und der von ihm benutzten Anwendung. Er beschreibt einen Ablauf, der als Folge einer Benutzeranfrage ausgeführt wird und bestimmten Bedingungen folgt. Eine Vorbedingung (engl.: precondition) muss erfüllt sein, damit eine Benutzeranfrage gestellt werden kann. Eine Nachbedingung (engl.: post condition) ist erfüllt, nachdem die Anwendung den regulären Fall des Ablaufs ausgeführt hat. Neben dem regulären Ablauf können Alternativen beschrieben werden, die beispielsweise im Fehlerfall ausgeführt werden.

Anwendungsfälle können in Textform oder in Form von grafischen Abläufen dokumentiert werden. In dieser Arbeit werden die regulären Abläufe durch die Geschäftsprozesse in Abschnitt 3.4 beschrieben. Um Wiederholungen zu vermeiden, sind die dort aufgeführten Abläufe in diesem Abschnitt nicht näher beschrieben. Die Beschreibungen der Anwendungsfälle erfolgen in Tabelle 3 bis Tabelle 10 und in englischer Sprache, um Konsistenz mit den Prozessbeschreibungen und der Implementierung zu ermöglichen.

Use Case	Login user
Actor	Dealer, Sales Admin
Precondition	None
Regular Sequence	1. User opens the web browser and navigates to the login page 2. User enters id and password
Post Condition	• User is logged in • The start page is displayed
Alternative Sequence	If the id is unknown, or the password is incorrect, the application does not log in the user and shows an error message.

Tabelle 3: Anwendungsfall Benutzer anmelden

Use Case	Qualify customer
Actor	Customer, Dealer
Precondition	• Dealer is logged in • The “qualify customer” page is displayed
Regular Sequence	See section 3.4
Post Condition	• Customer is identified • All necessary Customer data is available • The “select vehicle” page is displayed
Alternative Sequence	If the user cancels the operation, the cached data will be deleted and the start page is displayed.

Tabelle 4: Anwendungsfall Kunde identifizieren

Use Case	Select vehicle
Actor	Dealer
Precondition	• Dealer is logged in • The “select vehicle” page is displayed
Regular Sequence	See section 3.4
Post Condition	• A vehicle has been selected • All necessary vehicle data is available • The “create offer” page is displayed
Alternative Sequence	If the user cancels the operation, the cached data will be deleted and the start page is displayed.

Tabelle 5: Anwendungsfall Fahrzeug auswählen

Use Case	Create offer
Actor	Dealer
Precondition	• Dealer is logged in • The “create offer” page is displayed
Regular Sequence	See section 3.4
Post Condition	• The vehicle price is available • The service price(s) are available • The overall price is available • The financing mode is available • The confirmation page is displayed
Alternative Sequence	If the user cancels the operation, the cached data will be deleted and the start page is displayed

Tabelle 6: Anwendungsfall Angebot erstellen

Use Case	Send order
Actor	Dealer
Precondition	• Dealer is logged in • The “send order” page is displayed • At least one offer is available
Regular Sequence	See section 3.4
Post Condition	• The offer is converted to an order and has status confirmed • The order has been send to the Sales Admin • The start page is displayed
Alternative Sequence	If the user cancels the operation, the cached data will be deleted and the start page is displayed.

Tabelle 7: Anwendungsfall Bestellung verschicken

Use Case	Check order
Actor	Sales Admin
Precondition	• Sales Admin is logged in • An order has been received
Regular Sequence	See section 3.4
Post Condition	• On accept: the order has been accepted • On reject: the order has been rejected • A notification is sent to the Dealer
Alternative Sequence	If the user cancels the operation, the cached data will be deleted and the start page is displayed

Tabelle 8: Anwendungsfall Bestellung prüfen

Use Case	Manage offers
Actor	Dealer
Precondition	• Dealer is logged in • The “manage offers” page is displayed • At least one offer is available • Dealer is logged in and has sufficient rights to administrate the offer
Regular Sequence	See section 3.4
Post Condition	• On change: the offer is changed • On delete: the offer is deleted
Alternative Sequence	If the user cancels the operation, the cached data will be deleted and the start page is displayed.

Tabelle 9: Anwendungsfall Angebote verwalten

Use Case	Manage orders
Actor	Dealer
Precondition	• The “manage orders” page is displayed • At least one order is available • Dealer is logged in and has sufficient rights to administrate the order
Regular Sequence	See section 3.4
Post Condition	• On change: the offer is changed • On delete: order has been transferred back to an offer and is deleted
Alternative Sequence	If the user cancels the operation, the cached data will be deleted and the start page is displayed.

Tabelle 10: Anwendungsfall Bestellungen verwalten

Fahrzeuge, Händler und Benutzer werden von einem Administrator angelegt und verwaltet. Um den Entwicklungsaufwand im Rahmen des Machbaren zu belassen geschieht dies manuell in der Datenbank und ist nicht Teil des Prototyps. Die entsprechenden Anwendungsfälle sind daher in diesem Abschnitt nicht näher beschrieben. Wie die Daten in der Datenbank abgelegt werden, beschreibt der nächste Abschnitt.

3.3. Datenmodell

Die anfallenden Daten werden typischerweise in einer Datenbank gespeichert. Abbildung 12 stellt das Datenmodell der Anwendung als Entity-Relationship (ER) Diagramm in Krähenfußnotation [TEO05] dar. Sie zeigt die Entitäten für Kunde, Geschäftsdokumente, Händler, Fahrzeuge mit Sonderausstattungen und Dienstleistungen. Außerdem werden die Kardinalitäten dargestellt. Beispielsweise können einem Kunden kein, ein oder mehrere Geschäftsdokumente zugewiesen werden. Zwischen den Entitäten Kunde und Geschäftsdokument besteht damit eine 1..n Beziehung.

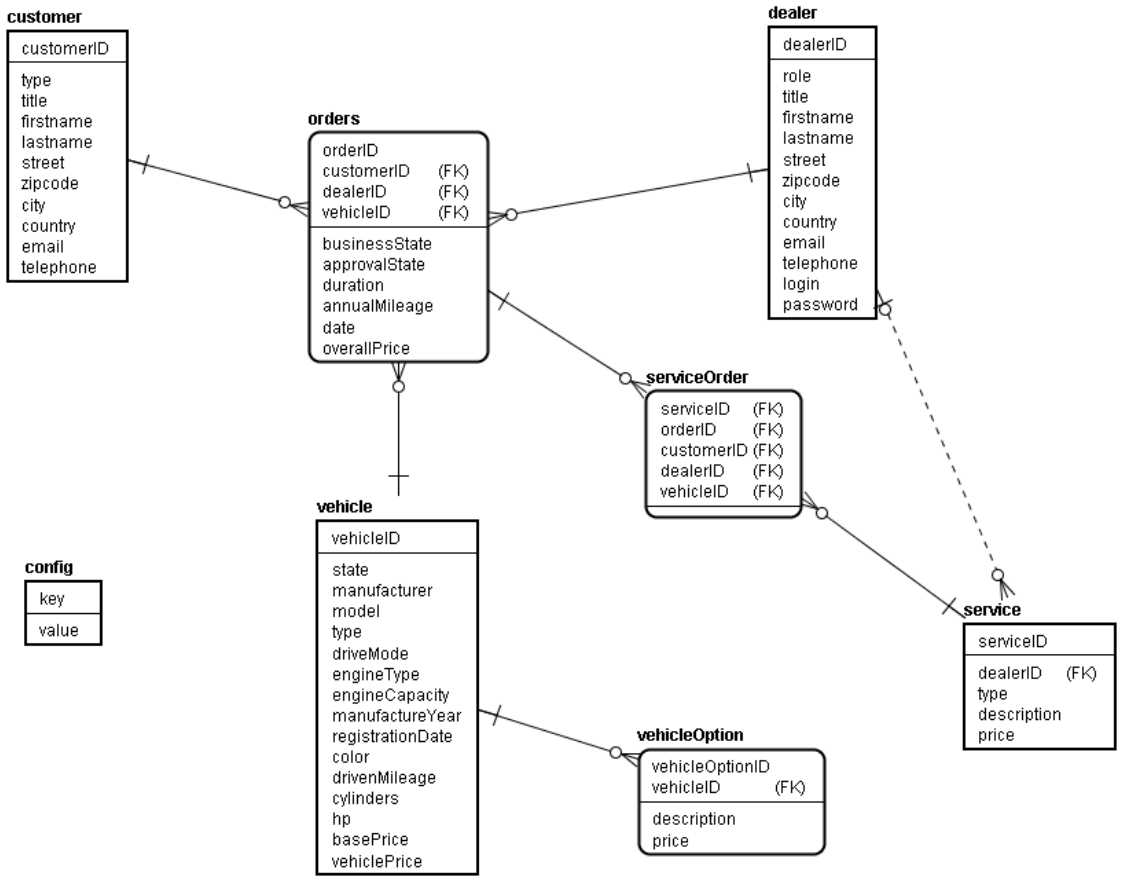

Abbildung 12: Datenmodell

3.3.1. Kunden und Händler

Kunden werden eine Adresse und ein Typ zugeordnet, der anzeigt ob der Kunde gewerblich (engl.: trade) oder privat (engl.: private) ist. Die Wertemenge für das Feld „type“ ist damit {private, trade}. Kunden werden durch eine Kundennummer eindeutig identifiziert. In der Tabelle ist die Kundennummer der Primärschlüssel. Händler haben neben den Adressdaten noch eine Rolle, die anzeigt ob sie als Händler oder Disponent mit dem System interagieren. Außerdem besitzen sie einen Benutzernamen und ein Passwort, um sich anmelden zu können und eine Händlernummer, die sie eindeutig identifiziert.

3.3.2. Fahrzeuge und Sonderausstattungen

Fahrzeuge haben einen Status (neu oder gebraucht), sind von einem bestimmten Modell und wurden von einem Hersteller in einem bestimmten Jahr produziert. Der Fahrzeugtyp beschreibt, ob es sich beispielsweise um ein Cabriolet oder einen Kleinbus handelt. Mögliche Fahrzeugtypen sind convertible, luxury, minivan, passenger, suv, sports, truck, van und wagon. Die Art der Schaltung (manual oder automatic) wird „drive mode“ bezeichnet. Der Motortyp bezeichnet die Kraftstoffart, mit der das Fahrzeug angetrieben wird. Mögliche Kraftstoffarten sind fuel, diesel, electric, LPG, biodiesel und hybrid. Der Hubraum (engl.: engine capacity) wird in Kubikzentimetern und die Leistung in Pferdestärken (engl.: horse power, kurz: hp) angegeben. Das Datum der Erstzulassung wird im Feld „registration date“ dokumentiert. Außerdem werden der Kilometerstand (engl.: driven mileage) und die Anzahl der Zylinder (engl.: cylinders) angegeben. Die in Abschnitt 3.1.2 eingeführten Basis- und Fahrzeugpreise werden direkt in der Fahrzeugtabelle abgespeichert.

Einem Fahrzeug können mehrere Sonderausstattungen (engl.: vehicle option) zugewiesen werden, die eine Bezeichnung und einen Preis haben. Der Fahrzeugpreis errechnet sich aus dem Basispreis und der Summe der Preise für Sonderausstattungen. Das Fahrzeug identifiziert die Sonderausstattung, d.h. es gibt keine Sonderausstattung, die keinem Fahrzeug zugewiesen ist. Dies wird durch die abgerundeten Ecken der Entität „vehicleOption“ in Abbildung 12 dargestellt. Man nennt solche Entitäten auch „schwache Entitäten“ [TEO05]. Auch Geschäftsdokumente und Dienstleistungsbestellungen sind schwache Entitäten.

3.3.3. Dienstleistungen und Geschäftsdokumente

Dienstleistungen können von Händlern angeboten werden, müssen es aber nicht. Beispielsweise wird eine Finanzierung häufig nicht von einem Händler sondern von einer Bank angeboten. Dienstleistungen sind einer Dienstleistungsklasse zugeordnet, die im Feld „type“ abgelegt werden. Mögliche Dienstleistungsklassen sind financing, leasing und service. Dienstleistungen enthalten außerdem eine Beschreibung und einen Preis.

Die Beziehung zwischen Dienstleistungen und Geschäftsdokumenten stellt die schwache Entität Dienstleistungsbestellung (engl.: service order) her. Sie ermöglicht, dass einer Dienstleistung mehrere Bestellungen und einer Bestellung mehrere Dienstleistungen zugeordnet werden können. Die stellt also eine n..m Beziehung dar.

Zu einem Geschäftsdokument gehören immer jeweils genau ein Kunde, ein Händler und ein Fahrzeug sowie beliebig viele Dienstleistungen. Mit dem Geschäftsdokument werden die in Abschnitt 3.1.2 beschriebenen Geschäfts- (offer, order oder contract) und Bewilligungsstatus (init, confirmed, accepted oder rejected) hinterlegt. Der Eintrag „duration“ bezeichnet die voraussichtliche Dauer der Geschäftsbeziehung, was beispielsweise für ein Leasing von Bedeutung ist. Auch die jährliche Laufleistung („annualMileage“) spielt in diesem Fall eine Rolle. Bei Änderung des Geschäftszustandes wird das Datum hinterlegt. Wenn alle nötigen Werte vorhanden sind, wird der Gesamtpreis berechnet. Der Basis- und Fahrzeugpreis ist dem zugeordneten Fahrzeug zu entnehmen.

Die Tabelle „config“ dient zum Ablegen von Konfigurationsdaten für die Anwendung, beispielsweise Mehrwertsteuersätze.

Das in dieser Arbeit beschriebene Datenmodell und die Geschäftsprozessmodelle sind aus Machbarkeitsgründen im Vergleich zur Praxis stark vereinfacht. Dadurch werden beispielsweise die Rabatte, die der Hersteller einem Händler gewährt, nicht modelliert. Die Geschäftsprozessmodelle werden im nächsten Abschnitt beschrieben.

3.4. Spezifikation der fachlichen Prozesse

3.4.1. Vorbemerkungen

Bei der Modellierung von Geschäftsprozessen wird zunächst die Wertschöpfungskette eines Unternehmens beschrieben, die als Überblick über den Ablauf dienen soll. Die Wertschöpfungskette eines Autoverkaufs geht vom ersten Kundenkontakt über die Bestellung und Produktion des Fahrzeugs bis hin zur Auslieferung und Garantieabwicklung. Im Fokus dieser Arbeit steht die Wertschöpfungskette vom Kundenkontakt bis zur Bestellung des Fahrzeugs. Die Elemente der Wertschöpfungskette werden als Geschäftsprozesse angesehen. Häufig werden dabei mehrere Ebenen verwendet, um die Darstellung übersichtlich halten zu können. Damit entstehen Unterprozesse, die Teil eines anderen sind.

Wie bei der Modellierung einer Produktlinie wird in dieser Arbeit zunächst ein Template der Geschäftsprozesse erstellt, dem alle Prozessbeteiligten folgen. Dieses Template wird auch „Blueprint“ genannt. Länderspezifische Abweichungen sind nicht Teil des Templates. Es ist jedoch sinnvoll, variable Teile zu kennzeichnen, um die Modellierung der Variabilitäten passend zum Template zu gestalten.

Die in diesem Abschnitt beschriebenen Geschäftsprozesse werden stellvertretend für einen Geschäftsanalysten mit dem IBM WebSphere Business Modeler (WBM) modelliert. Die im WBM verwendete Notation ist BPMN [OMG1] sehr ähnlich. Um die Überführung in Workflows zu ermöglichen, sind einige BPMN Elemente wie beispielsweise Ereignisse nicht verwendbar. Andere Elemente wie Human Tasks wurden dem Modell hinzugefügt, um die Modellierung von Benutzerinteraktionen zu ermöglichen. Geschäftsprozesse sind eine Aneinanderreihung von Aktivitäten, die in einer bestimmten Reihenfolge oder parallel abgearbeitet werden. Abbildung 13 stellt die in dieser Arbeit verwendete Notation dar.

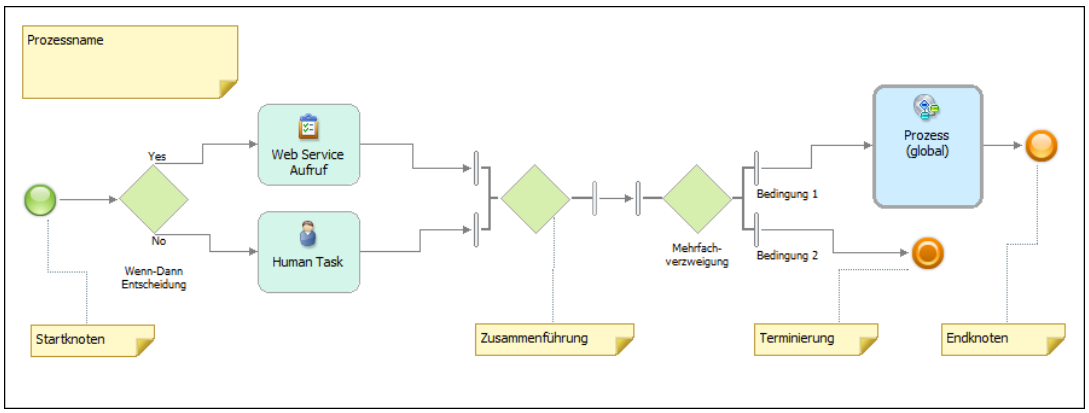

Abbildung 13: Legende Geschäftsprozesse

In BPMN werden Geschäftsprozesse in Form eines Graphen dargestellt. Die Knoten des Graphen heißen Aktivitäten, die entlang von gerichteten Kanten ausgeführt werden. Der Startknoten markiert den Beginn und der Endknoten das Ziel des Geschäftsprozesses. Bei einer „Terminierung“ wird ein Geschäftsprozess nicht erfolgreich abgeschlossen und ein Verwaltungseingriff wird nötig. Unterprozesse müssen weder Start- noch Endknoten haben, da sie wie Aktivitäten behandelt werden. Letztere können durch eine Software (Web Service Aufruf) oder einen Benutzer (Human Task) bearbeitet werden. Sie erhalten Eingabedaten und produzieren Ausgabedaten, mit deren Typ die jeweilige Kante beschriftet wird.

Im Hinblick auf die Implementierung dürfen Aktivitäten in der verwendeten Modellierungsumgebung nur eine ausgehende Kante haben. Wird die Aktivität beispielsweise in Java entwickelt, dann existiert nur ein Rückgabewert. Wenn mehrere Rückgabewerte benötigt werden, dann müssen diese in einem neuen, komplexen Datentyp zusammengefasst werden. Der Ablauf eines Geschäftsprozesses kann jedoch Verzweigungen haben, denen eine Bedingung zugeordnet ist. Ist Letztere eine exklusive ODER Bedingung, so folgt der Ablauf nur der ausgehenden Kante, für die die Bedingung zutrifft. Wenn-Dann Entscheidungen haben zwei und Mehrfachverzweigungen können mehr als zwei ausgehende Kanten haben, die mit einer Zusammenführung wieder vereinigt werden. Globale Elemente können außerhalb des aktuellen Geschäftsprozesses wiederverwendet werden. Der folgende Abschnitt beschreibt die fachlichen Prozesse zur Erstellung eines Angebots.

3.4.2. Angebotsprozess

Um ein Angebot zu erstellen, werden ein Kunde, ein Fahrzeug und ein Händler benötigt. Letzterer ist der Anwendung als angemeldeter Benutzer bekannt. Der Geschäftsprozess zur Angebotserstellung wird in Abbildung 14 dargestellt. Dieser besteht aus drei Unterprozessen: Kunde identifizieren (engl.: qualify customer), Fahrzeug auswählen (engl.: select vehicle) und Angebot erstellen (engl.: create offer).

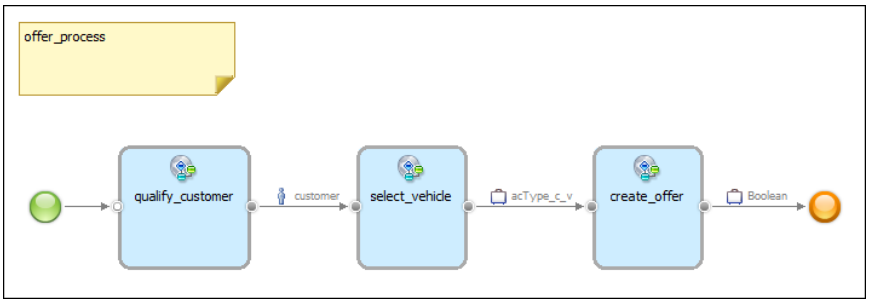

Abbildung 14: Angebotsprozess

Zunächst muss der Kunde identifiziert werden. Dies geschieht typischerweise am Anfang eines Gespräches zwischen Kunde und Händler. Das hat den Vorteil, dass zuerst die Kundendaten erfasst werden, die für das Marketing möglicherweise von Nutzen sind. Am Ende des in Abbildung 15 dargestellten Unterprozesses sind, wie in Abschnitt 3.2.2 beschrieben, alle Kundendaten bekannt, d.h. die Ausgangsdaten sind vom Typ Kunde. Diese sind Eingangsdaten für den darauffolgenden Unterprozess „Fahrzeug auswählen“. Ist das Fahrzeug gewählt, so werden Kunde und Fahrzeug zu einem komplexen Datentyp (`acType_c_v`) zusammengefasst. Damit kann schließlich ein Angebot erstellt werden.

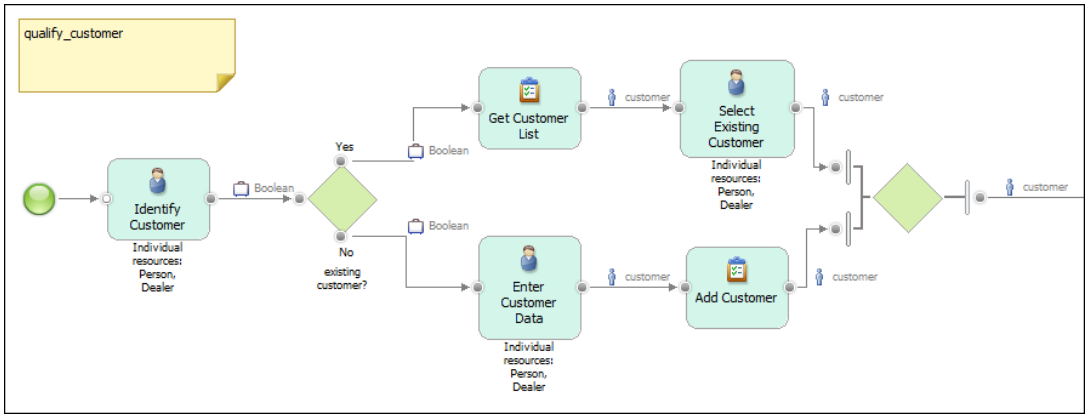

Abbildung 15: Geschäftsprozess Kunde identifizieren

Der Unterprozess „Kunde identifizieren“ benötigt keine Eingangsdaten. Zu Beginn muss vom Benutzer entschieden werden, ob der Kunde Neu- oder Bestandskunde ist. Dies entspricht einer Wenn-Dann Entscheidung. Wenn der Kunde bereits im Kundenbestand ist, dann wird eine Liste aller Kunden angezeigt, aus der der richtige Kunde ausgewählt wird. Im Falle eines Neukunden müssen die Daten des Kunden vom Händler

eingegeben werden. Diese Eingaben werden direkt im Anschluss in die Datenbank geschrieben. Welcher Weg auch gewählt wurde, am Ende sind die Kundendaten bekannt und die Auswahl des Fahrzeugs, die in Abbildung 16 dargestellt ist, kann beginnen.

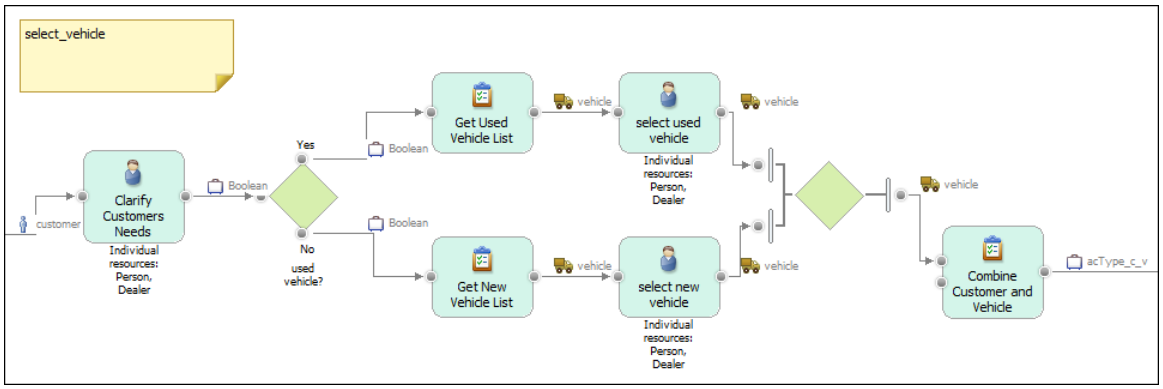

Abbildung 16: Geschäftsprozess Fahrzeug auswählen

Um ein Fahrzeug auswählen zu können, fragt der Händler den Kunden zunächst, ob er ein Angebot für ein Gebraucht- oder ein Neufahrzeug haben möchte. Gemäß dieser Auswahl bekommt er eine Liste von Fahrzeugen angezeigt, aus denen er eines ausgewählt. Am Ende muss aus den Daten von Kunde und Fahrzeug ein komplexer Datentyp erstellt werden, da Beides an den Unterprozess zur Angebotserstellung (Abbildung 17) übergeben wird. Die Aktivität „Combine Customer and Vehicle“ in Abbildung 17 hat zwei Eingänge. Der Zweite ist vom Typ Kunde, dessen Eingangsdaten innerhalb des Geschäftsprozesses bereits bekannt sind. Aus Darstellungsgründen wird auf eine Verzweigung des Datenflusses im Geschäftsprozess verzichtet.

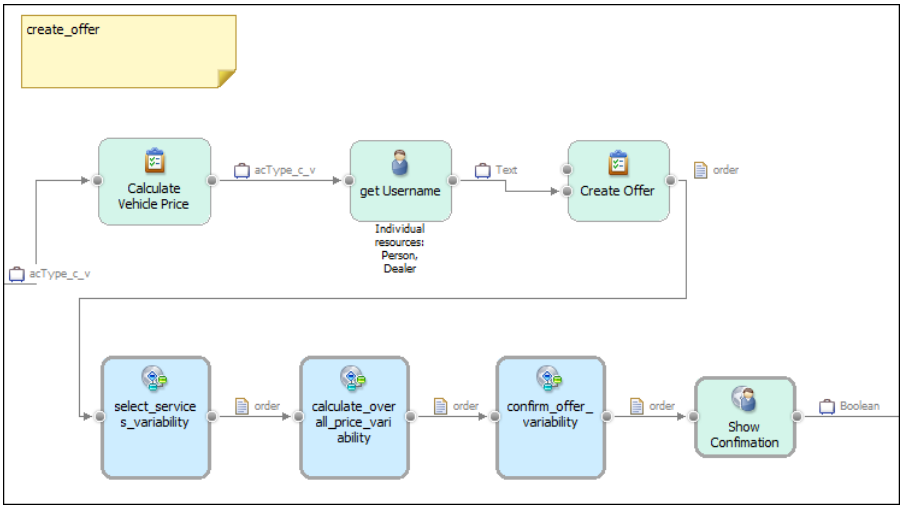

Abbildung 17: Geschäftsprozess Angebot erstellen

Anhand des Basispreises und den Sonderausstattungen des gewählten Fahrzeugs wird der Fahrzeugpreis berechnet. Die darauffolgende Aktivität benötigt keine Interaktion mit dem Benutzer. Stattdessen wird anhand der Anmeldeinformationen des Benutzers der Händler ermittelt. Letzterer wird mit in das zu erstellende Angebot eingetragen, welches direkt nach der Erstellung in die Datenbank geschrieben wird. Nach der Auswahl der Dienstleistungen wird der Gesamtpreis berechnet und das Angebot bestätigt. Diese Aktivitäten sind variabel und in Abschnitt 3.5 beschrieben. Am Ende eines jeden Prozesses erhält der Benutzer eine Bestätigungsmeldung, die global modelliert ist, um von anderen Prozessen wiederverwendet werden zu können.

Nach Erstellung eines Angebots kann dieses Angebot in eine Bestellung umgewandelt werden. Dieser Prozess ist im nächsten Abschnitt beschrieben.

3.4.3. Bestellprozess

Viele Kunden lassen sich mehrere Angebote erstellen, bevor sie sich für eine Bestellung entscheiden. Es ist daher sinnvoll, die Angebots- und Bestellprozesse voneinander zu trennen. Letzterer ist in Abbildung 18 dargestellt.

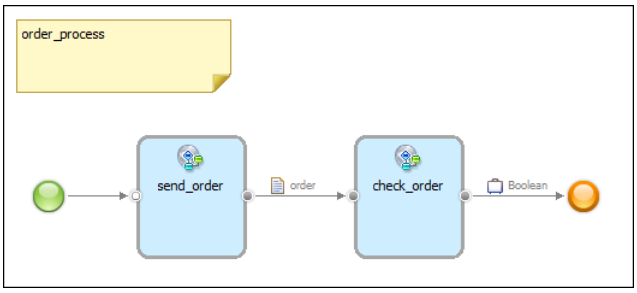

Abbildung 18: Bestellprozess

Der Bestellprozess besteht aus zwei Teilen. Zunächst wird eine Bestellung an den zuständigen Disponenten verschickt, der diese auf ihre Richtigkeit hin überprüft. Der Geschäftsprozess zum Versand der Bestellung ist in Abbildung 19 dargestellt.

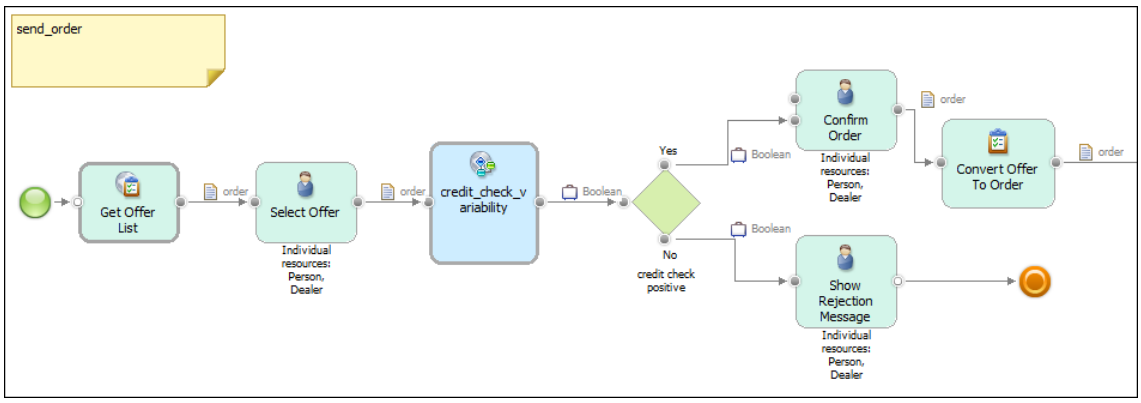

Abbildung 19: Geschäftsprozess Bestellung verschicken

Bevor eine Bestellung verschickt werden kann, muss sie zu einer solchen gemacht werden. Dafür wählt der Händler ein Angebot aus, das in eine Bestellung umgewandelt werden soll. Danach findet eine Prüfung der Kreditwürdigkeit des Kunden statt. Ist das Ergebnis der Kreditprüfung negativ, so wird eine der Grund für die Ablehnung angezeigt und der Prozess terminiert. Ist ihr Ergebnis positiv, so wird der Händler aufgefordert, die Bestellung zu bestätigen. Eine Bestätigung kann nur dann erfolgen, wenn der Kunde die Bestellung unterschrieben hat. Nach der Bestätigung wird in der Datenbank aus dem Angebot eine Bestellung mit Status „bestätigt“ und sie wird zur Prüfung weitergegeben.

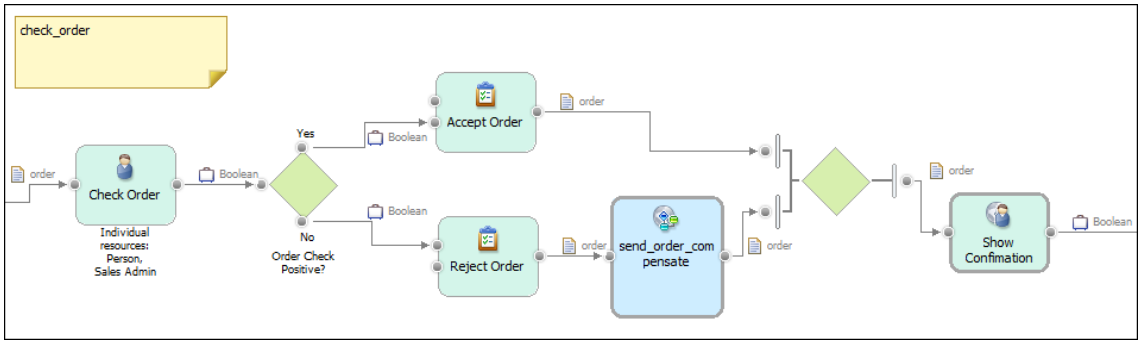

Abbildung 20: Geschäftsprozess Bestellung prüfen

Wie in Abbildung 20 gezeigt, prüft ein Disponent die Bestellung. Dazu gehören ein manueller Plausibilitätscheck und ein Abgleich mit den Bestellkontingenten des Händlers. Hat der Händler beispielsweise sein Kontingent überschritten, so wird die Bestellung abgelehnt und kompensiert. Die Kompensation bewirkt, dass die Bestellung in der Datenbank wieder zurück in ein Angebot umgewandelt wird. Der Händler kann dieses Angebot zu einem späteren Zeitpunkt zurück in eine Bestellung umwandeln. Ist mit der Bestellung alles in Ordnung, dann akzeptiert der Disponent die Bestellung, der Status „akzeptiert“ wird in der Datenbank gesetzt und eine Bestätigung wird angezeigt.

Der folgende Abschnitt beschreibt, wie einmal erstellte Anwendungen nachträglich bearbeitet werden können.

3.4.4. Verwaltungsprozesse

Damit Angebote nicht jedes Mal neu erstellt werden müssen, können sie bearbeitet werden. Sollten sie nicht mehr benötigt werden, ist es möglich, sie zu löschen. Diesen Zwecken dient der in Abbildung 21 dargestellte Geschäftsprozess.

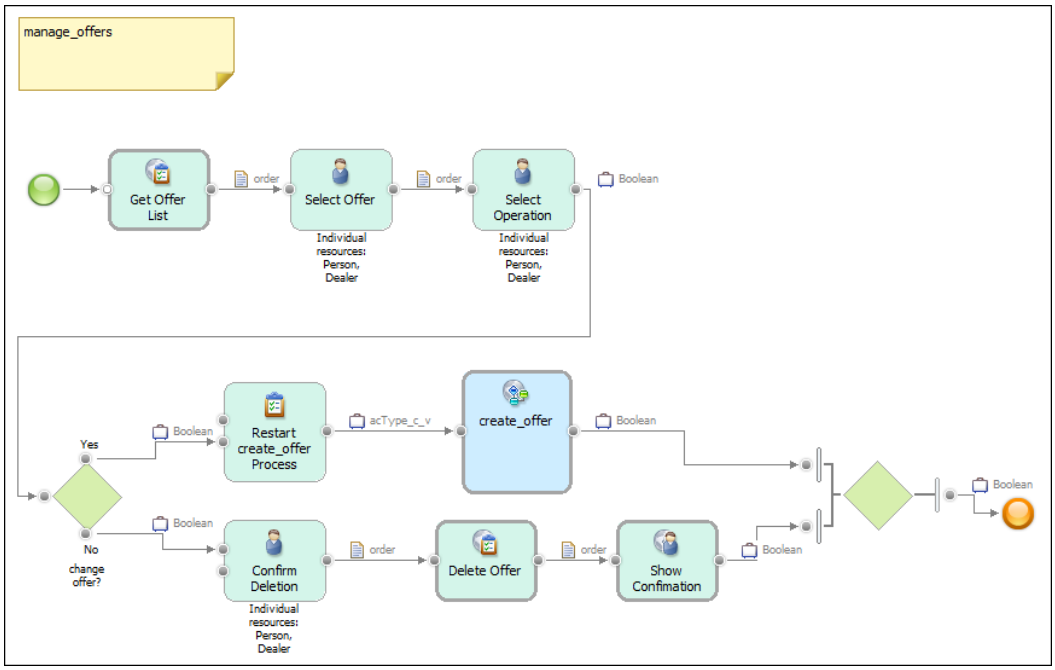

Abbildung 21: Geschäftsprozess Angebote verwalten

Zur Auswahl eines Angebots wird derselbe Dienst wie in Abbildung 19 verwendet. Nachdem der Händler ein Angebot ausgewählt hat, muss er festlegen, ob er es bearbeiten oder löschen möchte. Die Bearbeitung wird technisch durch Löschen und erneutes Erstellen des Angebots in der Datenbank gelöst. Häufig ändern sich bei der Bearbeitung weder der Kunde noch das Fahrzeug, weshalb diese im Fall einer Bearbeitung bestehen bleiben. Nur der in Abbildung 17 gezeigte Geschäftsprozess wird erneut durchlaufen und an dieser Stelle wiederverwendet. Sollten sich Kunde oder Fahrzeug ändern, muss der Angebotsprozess von Anfang an durchlaufen werden. Soll das Angebot gelöscht werden, muss der Händler dies zunächst bestätigen. Am Ende wird die bereits mehrfach verwendete Bestätigung angezeigt.

Die in Abbildung 22 dargestellte Verwaltung von Bestellungen unterscheidet sich in mehreren Punkten von der Angebotsverwaltung.

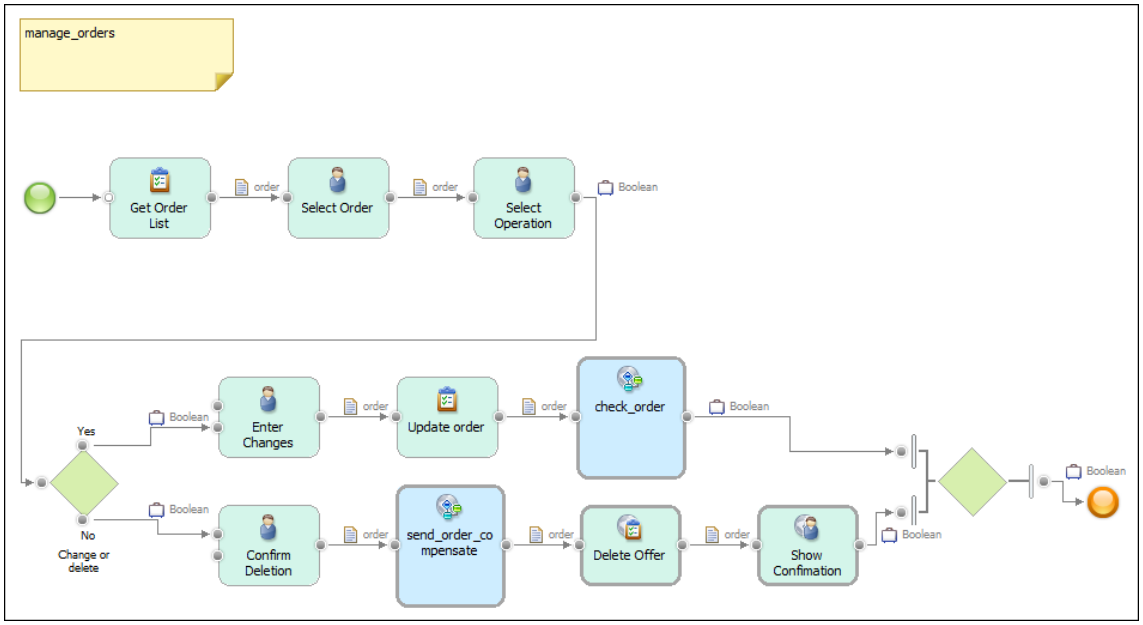

Abbildung 22: Geschäftsprozess Bestellungen verwalten

Zu Beginn müssen eine Bestellung und die entsprechende Operation ausgewählt werden. Sowohl ein Händler als auch ein Disponent können Bestellungen verwalten. Die benötigte Rolle ist deshalb in Abbildung 22 nicht angegeben. Grund für eine Änderung ist beispielsweise die Verlängerung der Vertragslaufzeit bei Leasingverträgen. In jedem Fall muss die Bestellung abschließend erneut von einem Disponenten geprüft werden. Eine direkte Löschung einer Bestellung ist nicht möglich. Stattdessen muss die Bestellung zunächst in ein Angebot zurückgewandelt werden, um gelöscht werden zu können.

Die in Abschnitt 3.4 beschrieben Prozesse sind für alle Märkte gleich. Marktspezifisches Verhalten ist im folgenden Abschnitt beschrieben.

3.5. Beschreibung der Variabilitäten

3.5.1. Fachliche Variabilitäten

Mehrere Teile der Geschäftsprozesse sind abhängig vom Kontext, in dem sie ausgeführt werden. Abbildung 17 und Abbildung 19 enthalten variable Aktivitäten, die durch globale Unterprozesse weiter verfeinert werden.

Die Menge der auswählbaren Dienstleistungen ist abhängig vom Fahrzeugtyp, d.h. bei Gebrauchtfahrzeugen ist keine Garantieverlängerung möglich. Auch der Preis für eine Dienstleistung ist abhängig vom Händler, der sie anbietet. Der Gesamtpreis eines Geschäftsdokumentes ist abhängig vom Land des Händlers, da in jedem Land ein anderer Mehrwertsteuersatz anzurechnen ist.

Im Markt Spanien muss ein Kunde nicht nur eine Bestellung, sondern auch ein Angebot als Absichtserklärung unterschreiben. Damit ist bei allen Benutzern mit Händlerland Spanien eine zusätzliche Aktivität notwendig. Die Prüfung der Kreditwürdigkeit eines Kunden ist abhängig vom Land, in dem er wohnt, d.h. je nach Herkunftsland des Kunden muss ein anderer Service aufgerufen werden.

3.5.2. Modellierung der Variabilitäten

Da die Variabilitäten einen fachlichen Bezug haben, ist es sinnvoll, diese in Form von fachlichen Unterprozessen zu modellieren. Alle (nicht globalen) Aktivitäten eines solchen Unterprozesses haben einen eindeutigen Bezug zu diesem, d.h. alle Variabilitäten sind leicht identifizierbar und explizit modelliert. Der Unterprozess ist damit dem in Abschnitt 2.3.4 beschriebenen Konzept der Variabilitätspunkte sehr ähnlich. Typischerweise wird ein Prozessmodell mit einem Gültigkeitsdatum versehen. Wird eine neue Variante des Prozessmodells erstellt, so gilt diese für alle ab diesem Zeitpunkt erstellten Prozessinstanzen, nicht aber für bereits Laufende. Dies ermöglicht das nachträgliche Austauschen, ohne die übergeordneten Prozesse ändern zu müssen.

Die ersten beiden Variabilitäten betreffen nur die Daten, nicht aber die Abläufe des Geschäftsprozesses. Abbildung 23 zeigt die Variabilität zur Auswahl von Dienstleistungen.

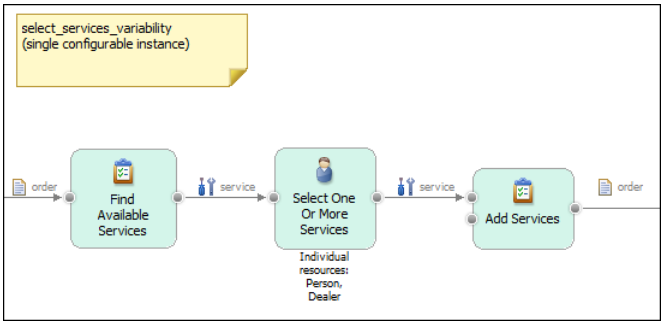

Abbildung 23: Variabilität Dienstleistungen auswählen

Für das Erstellen einer Auswahl ist der Kontext des Angebots zu beachten. Das Fahrzeug ist im Angebot hinterlegt. Steht der Fahrzeugstatus auf gebraucht, kann der Benutzer keine Garantieverlängerung auswählen. Auch der Händler ist im Angebot hinterlegt und kann Einfluss auf die Preise der Dienstleistungen haben. Händler-ID und Fahrzeugstatus bilden einen Mandantenkontext. Der Unterprozess implementiert damit das Service Pattern „Single Configurable Instance“ aus Abschnitt 2.3.3 und das zugehörige Invocation Pattern „Invocation under Tenant Context“. Nicht nur Unterprozesse, auch einzelne Aktivitäten können unter Mandantenkontext aufgerufen werden. Ein Beispiel dazu ist in Abbildung 25 dargestellt.

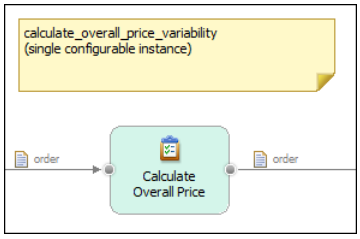

Abbildung 24: Variabilität Gesamtpreisberechnung

Dieser Unterprozess interagiert nicht mit dem Benutzer, kapselt jedoch die Single Configurable Instance Aktivität. Der übergeordnete Prozess muss bei einer Änderung der Berechnungsregeln nicht geändert werden. Dies entspricht dem „Tenant Context Wrapper“ Pattern. Unter bestimmten Voraussetzungen ist die Benutzung des „Multiple Intance“ Patterns empfehlenswert, beispielsweise wenn wie in Abbildung 25 abhängig vom Kontext ein variierender Prozessablauf entsteht.

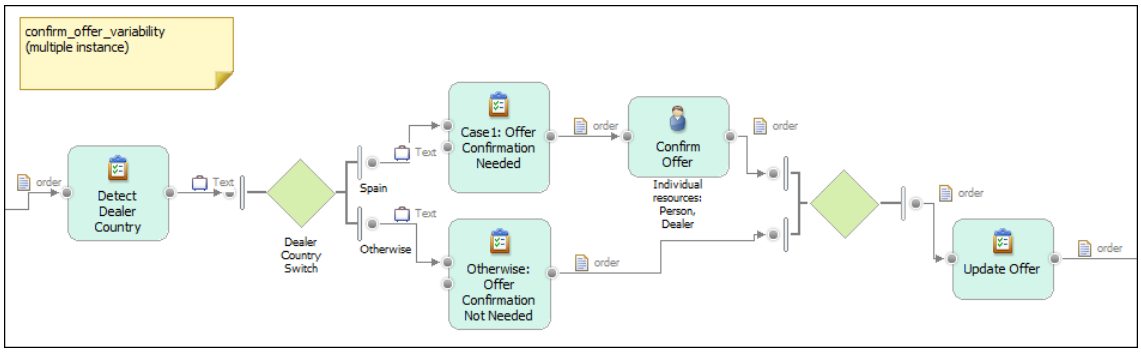

Abbildung 25: Variabilität Angebot bestätigen

Im Markt Spanien müssen die Kunden ein Angebot bestätigen, bevor es gültig wird. Dazu wird ein neuer kontextabhängiger Prozessschritt vom Benutzer gefordert. Als erste Aktivität wird der Mandantenkontext, d.h. das Händlerland extrahiert. Der Kontext wird demnach dem Prozessfluss hinzugefügt, was dem „Tenant Context Appender“ Pattern entspricht. Er dient als Grundlage für die darauffolgende Entscheidung, welcher Pfad, d.h. welche Folgeaktivitäten gewählt werden soll. Letztere gibt die für den Folgeprozess benötigten Daten weiter. Der Unterprozess „Angebot bestätigen“ als Ganzes ist ein Beispiel für das „Tenant Context Based Router“ Pattern aus Abschnitt 2.3.3. Für die Verwendung eines „Multiple Instance“ Patterns ist eine Benutzerinteraktion keine zwingende Voraussetzung. Wie Abbildung 26 zeigt, können auch Implementierungsgründe eine Rolle spielen.

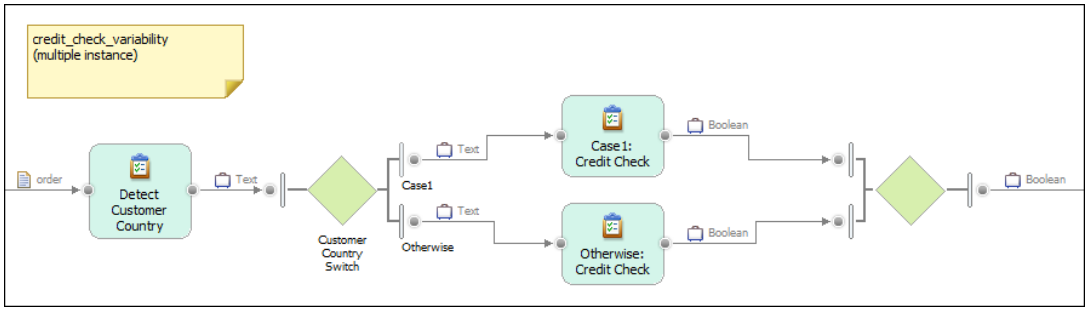

Abbildung 26: Variabilität Kreditwürdigkeit prüfen

Die Prüfung der Kreditwürdigkeit soll als Beispiel für einen internen Variabilitätspunkt dienen. Der Benutzer bemerkt die Unterschiede im internen Ablauf nicht. Tatsächlich wird je nach Herkunftsland des Kunden ein anderer Service zur Kreditprüfung verwendet. Alle in Abschnitt 2.3.3 beschriebenen Patterns werden durch die in diesem Abschnitt beschriebenen Beispiele abgedeckt.

Für eine vollständige Evaluierung reicht die Modellierung der Prozesse nicht aus, da unter Umständen für die Entwicklung wichtige Details vergessen werden können. Geschäftsprozesse müssen in ausführbare Workflows und Aktivitäten in Tasks bzw. Web Services umgewandelt werden. Letztere benötigen eine Implementierung, die im nächsten Kapitel beschrieben ist.

4. Implementierung

Dieses Kapitel beschreibt die Entwicklung des Prototyps. Abschnitt 4.1 beschreibt deren zugrundeliegende Architektur. Die Details der einzelnen Entwicklungsschritte sind in den darauffolgenden Abschnitten beschrieben.

4.1. Ablauf der Implementierung

4.1.1. Systemumgebung

In der Automobilindustrie empfiehlt sich die Verwendung einer kommerziellen Entwicklungsumgebung [IBM1], um beispielsweise vom besseren Support profitieren zu können. Marktführer im Bereich SOA [IBM7] ist der Soft- und Hardwarehersteller IBM, der auch in Verbindung mit BPM [IBM8] an vielen Standards mitgewirkt hat. Die Produkte der IBM WebSphere Familie wurden daher als Basis für die Softwareentwicklung im Kontext dieser Arbeit verwendet.

Für die Modellierung der in Abschnitt 3.4 beschriebenen Geschäftsprozesse wird der IBM WebSphere Business Modeler Advanced (WBM) in der Version 6.1.2 verwendet. Die dazu passende Software Entwicklungsumgebung ist der WebSphere Integration Developer (WID) 6.1.2 [IBM6], der eine Workflow Engine, den WebSphere Process Server (WPS) 6.1.2 [IBM5], bereits enthält. Letzterer enthält die Komponenten Enterprise Service Bus, eine Testumgebung und eine auf mehrere Server verteilbare Laufzeitumgebung (WebSphere Application Server Network Deployment 6.1), die eine einfache Skalierung der Anwendung ermöglicht. Um den Integrationsaufwand gering zu halten, ist das Datenbanksystem vom gleichen Hersteller, eine IBM DB2 Express-C 9.5 [IBM9]. Als Betriebssystem wird Microsoft Windows Vista Business SP1 verwendet; getestet wird zusätzlich unter Windows XP Professional SP3.

Für die Modellierung der Anwendungsfälle in Abschnitt 3.2 und des Datenmodells in Abschnitt 3.3 wird JUDE/Professional 5.4 [CHA1] verwendet, das zur Modellierung in UML verwendet wird. Die Versionsverwaltung erfolgt mithilfe von Subversion und TortoiseSVN [TIG1].

4.1.2. Entwicklungsprozess

Der Entwicklungsprozess der Software erfolgt iterativ und orientiert sich an dem in Abschnitt 2.2.2 beschriebenen Lebenszyklus eines Workflows. Zunächst werden die Geschäftsprozesse in BPMN modelliert. Daraus werden automatisch die entsprechenden Workflows generiert. Zeitgleich wird das Datenmodell als ER-Diagramm modelliert, aus dem die Datenbanktabellen generiert werden. Die generierten Workflows werden auf dem WPS installiert und ausgeführt. Mit der im WID integrierten Testumgebung werden sie getestet und im nächsten Zyklus verfeinert. Um einen Workflow testen zu können, müssen die aufgerufenen Web Services Werte zurückgeben, d.h. zumindest teilweise implementiert werden. Nachdem alle Workflows lauffähig sind, werden die Benutzeroberfläche und die Datenbank in die Anwendung integriert.

Die Anwendung wird also von innen nach außen entwickelt, was durch die im folgenden Abschnitt beschriebene Architektur ermöglicht wird.

4.1.3. Architekturüberblick

Die Anwendung ist in vier Schichten aufgeteilt, wobei jede Schicht Zugriff auf die Dienste der darunterliegenden Schicht erhält. Abbildung 27 stellt diese Schichten und die ihnen zugeordneten Laufzeitumgebungen schematisch dar. Die Architektur orientiert sich an der in Abschnitt 2.3.1 beschriebenen SaaS Architektur. Zur besseren Übersicht werden ebenenübergreifende Komponenten, wie Metadaten oder Sicherheitsdienste nicht in der Abbildung dargestellt.

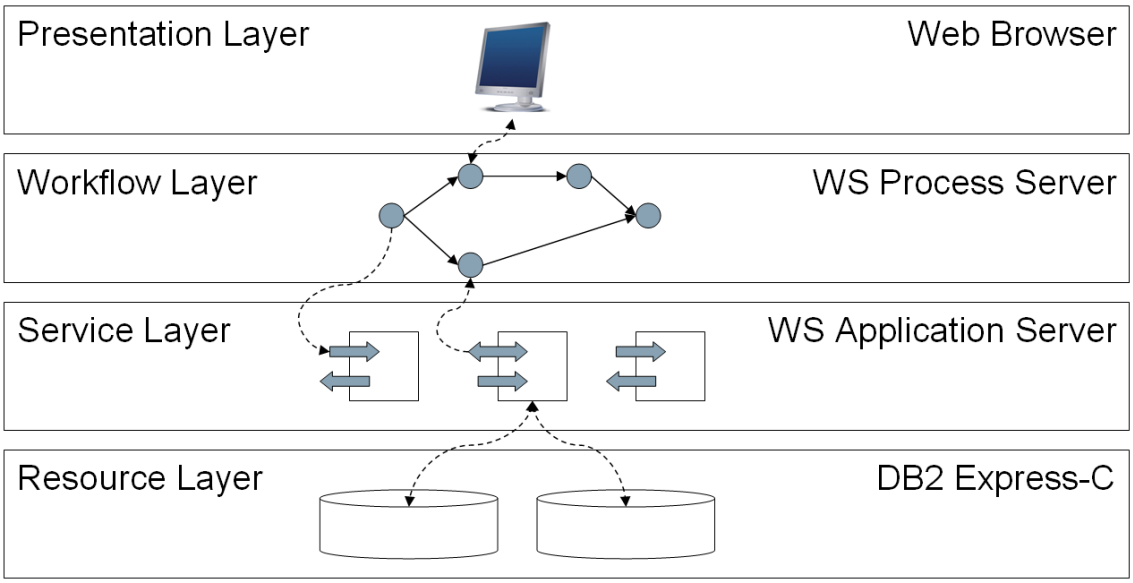

Abbildung 27: Architekturüberblick

Der Benutzer interagiert mit der Anwendung mithilfe eines Web Browsers und einer Benutzeroberfläche, die gemeinsam die Präsentationsebene bilden. Er erfüllt dabei genau die Tasks, die er als Prozessbeteiligter zu erfüllen hat. Die übrigen Aktivitäten des Workflows werden von Web Services ausgeführt, die in Java entwickelt sind. Einige dieser Web Services greifen dabei auf die Datenbank zu. Den Kern der Anwendung bilden damit die Workflow und die Service Ebene. Diese enthalten die komplette Geschäftslogik der Anwendung, an der sich die Benutzeroberfläche und die Datenbank orientieren.

4.2. Umsetzung des Geschäftsprozessmodells

4.2.1. Export des Geschäftsprozessmodells

Der WebSphere Business Modeler Advanced beinhaltet einen speziellen Modellierungsmodus namens „WebSphere Process Server“ [IBM14]. Dieser ermöglicht die Modellierung von Prozessmodellen, die direkt in auf dem WPS ausführbare Workflows exportiert werden können. Dabei werden Modellierungselemente, die nicht in BPEL abgebildet werden können, eingeschränkt. Wie in Abschnitt 2.2.3 beschrieben, existiert

beispielsweise kein BPEL Element für eine for-Schleife. Eine weitere wichtige Einschränkung ist, dass der Kontrollfluss im Gegensatz zu anderen Modellierungsmodi keine Zyklen enthalten darf. Letztere können stattdessen in Form von while-Schleifen modelliert werden, die eine festgelegte Abbruchbedingung und damit deterministisches Verhalten besitzen. Werden zwei Aktivitäten verbunden, so müssen Aus- und Eingabedaten denselben Typ und dieselbe Kardinalität besitzen. Dies gilt auch für die Ein- und Ausgabedaten der einzelnen Aktivitäten.

Der WBM unterstützt den Benutzer durch entsprechende Meldungen, wenn Fehler in der Modellierung erkannt werden. Dazu gehören auch benötigte Zusatzinformationen, z.B. müssen für jede Verzweigung und Schleife entsprechende Bedingungen definiert werden. Diese sind nötig, damit die Workflow Engine später erkennt, welchem Pfad im Workflow gefolgt werden soll. Eine Verzweigung kann auf zwei Arten exportiert werden: Entweder als Kante, der nur bei zutreffender Bedingung (engl.: transition condition) gefolgt wird, oder als `<switch>` Aktivität, in der Bedingungen in Form von Java Fragmenten definiert werden. Welche Art gewählt wird, wählt der Benutzer in den technischen Attributen, die nur im WPS Modus sichtbar sind. Dort können auch weitere Einstellungen vorgenommen werden, beispielsweise Namensräume oder die Implementierungsart eines Web Services.

Die Datentypen werden in Geschäftsobjekten definiert, aus denen beim Export XML Schema Definitionen (XSD) generiert werden. Einer Verbindungskante zwischen zwei Aktivitäten wird bei der Modellierung ein solches Geschäftsobjekt zugewiesen. Auf diese Weise wird der Datenfluss modelliert, der in BPEL allerdings nicht explizit existiert. Stattdessen wird der Datenfluss mithilfe des `<variables>` Elementes abgebildet. Die Ausgabevariable wird auf die Eingabevariable der Folgeaktivität abgebildet und beim Export werden die zu den Aktivitäten passenden WSDL Dateien exportiert. Die Benutzung von `<assign>` Aktivitäten zur Datenzuweisung entfällt, da der WPS automatisch die Werte an die Folgeaktivitäten weiterleitet.

Ein globaler Geschäftsprozess wird beim Export in einen BPEL Workflow (`<process>` mit untergeordnetem `<flow>` Element) umgewandelt, während ein lokaler Unterprozess in einer `<scope>` mit `<flow>` Aktivität zusammengefasst wird. Das hat zur Folge, dass ein lokaler Unterprozess nicht einzeln ausgetauscht werden kann, sondern nur zusammen mit dem Übergeordneten. Alle in Kapitel 3 beschriebenen Geschäftsprozesse sind daher global. Dies hat zwar Performanceeinbußen zur Folge, ermöglicht aber die Wiederverwendung und Versionsverwaltung der Workflows. Abbildung 28 zeigt beispielhaft den Workflow, der aus dem exportierten Geschäftsprozess zur Identifikation eines Kunden aus Abbildung 15 generiert wird.

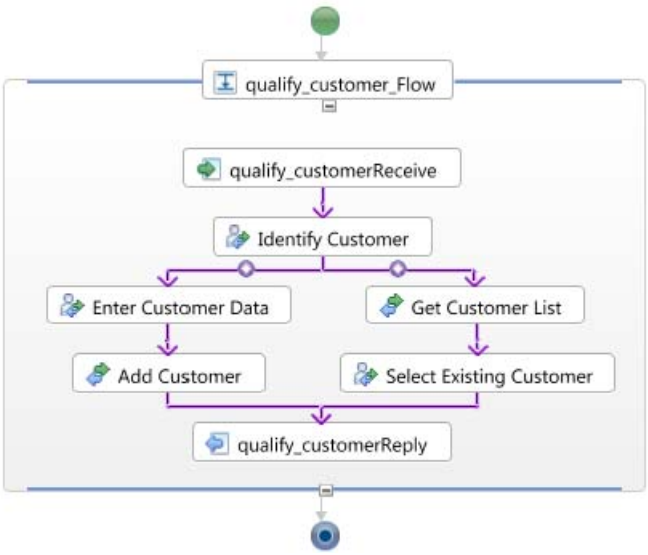

Abbildung 28: Workflow im WID

Aktivitäten werden abhängig von deren Modellierung als `<invoke>` Element exportiert. Ist in den technischen Attributen eine Implementierungsart, z.B. Java, ausgewählt, dann wird die entsprechende Javaklasse mit den entsprechenden Eingabeparametern erzeugt. Lokale Human Tasks werden direkt in den Workflow integriert, während globale Tasks als alleinstehende Elemente exportiert und wiederverwendet werden können. Rollen und die Zuweisung der Rollen zu den Tasks können nur dann exportiert werden, wenn das vordefinierte Template aus dem WBM verwendet wird. Wenn wie im Fokus dieser Arbeit ein eigenes Datenmodell für Benutzer und Kunden verwendet wird, muss die Rollenverteilung von der Anwendung gesteuert werden.

Alle Elemente gemeinsam werden in eine Archivdatei exportiert, die von der Zielanwendung dank einem Austauschformat importiert werden kann. Dies beschreibt der nächste Abschnitt.

4.2.2. Import der Workflows

Wird die Archivdatei aus dem WBM im WebSphere Integration Developer importiert, werden in der Geschäftsintegrationsansicht drei Module angezeigt. Das Workflowmodul enthält die generierten Workflows und das Implementierungsmodul die zu entwickelnden Web Services. Eine Bibliothek enthält die Datentypen und Schnittstellen, die von den beiden anderen Paketen benutzt werden.

Web Services, globale Workflows und Human Tasks werden, wie in Abschnitt 2.2.4 beschrieben zu Service Komponenten einer SCA abstrahiert. Diese Komponenten können grafisch verbunden werden. So genannte SCA-Imports ermöglichen den Zugriff auf Komponenten anderer Module, die einem SCA-Export zugewiesen sind. Abbildung 29 zeigt das SCA Diagramm für das Workflow Modul.

Abbildung 29: SCA-Diagramm

Alle Komponenten, die nach außen hin verfügbar sind, d.h. auf die Elemente der darüberliegenden Architekturschicht zugreifen dürfen sind mit einem SCA-Export gekennzeichnet. In Abbildung 29 sind dies die beiden Haupt- und die Verwaltungsprozesse. Auf die Elemente der darunterliegenden Ebene, d.h. auf die Web Services wird mit SCA-Imports zugegriffen. Auch die Workflows können im WID grafisch bearbeitet werden, allerdings orientiert sich die dortige Notation stärker an BPEL, so dass beispielsweise kein expliziter Datenfluss sichtbar ist.

Abläufe eines Unternehmens werden meist nicht durch die Softwareentwickler sondern durch die Fachabteilungen festgelegt. Dies bedeutet, dass nicht die Workflows, sondern die Geschäftsprozesse geändert werden. Allerdings unterscheiden sich die modellierbaren Elemente von WBM und WID stark. Technische Details, wie Fehlerbehandlungen oder der Aufruf von Web Services können nur im WID geändert werden. Die Folge sind Inkonsistenzen zwischen Geschäftsprozess und Workflow, die umso gravierender werden, je mehr die iterative Entwicklung einer Anwendung voranschreitet. Ändert sich der Geschäftsprozess, dann muss dieser erneut aus dem WBM exportiert und in den WID importiert werden. Damit dadurch nicht alle Änderungen erneut vorgenommen werden müssen, steht ein Synchronisierungsmechanismus zur Verfügung. Dieser ist in Abbildung 30 dargestellt.

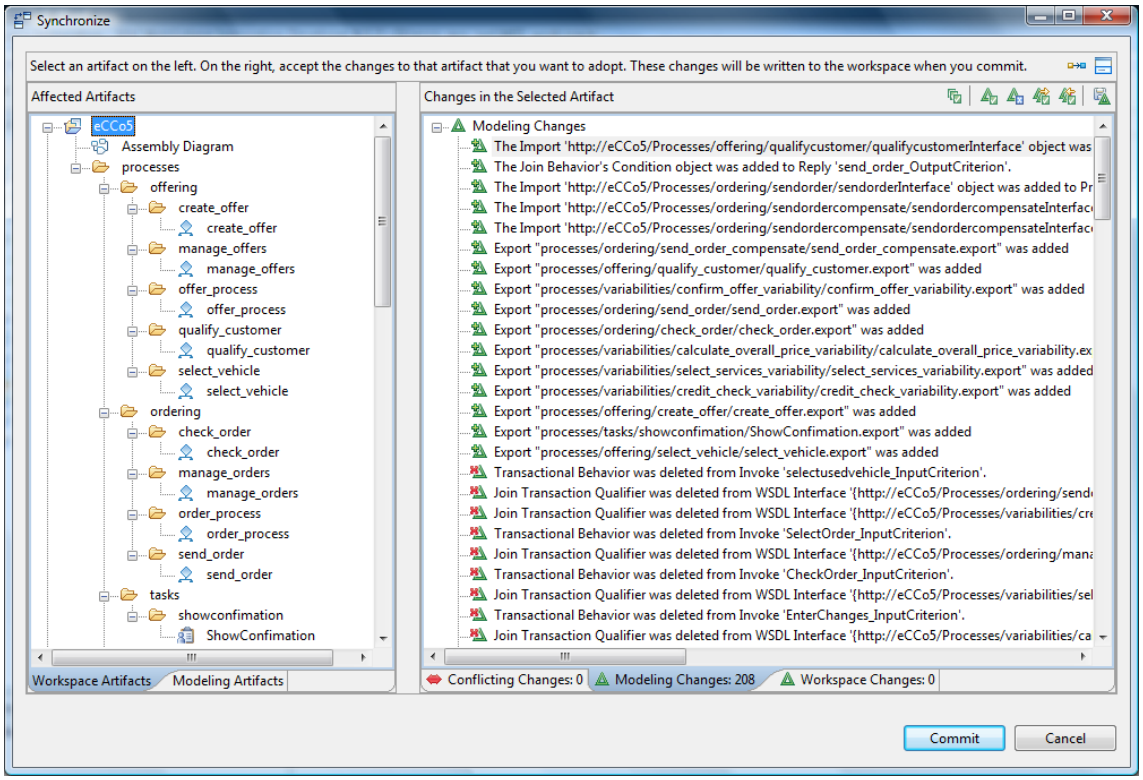

Abbildung 30: Synchronisierungsfenster

Bei der Synchronisierung werden alle geänderten Elemente angezeigt. Unterschieden wird dabei zwischen Änderungen am Geschäftsprozessmodell und Änderungen in der Entwicklungsumgebung. Wie Abbildung 30 zeigt, werden die Unterschiede schnell zahlreich und auch nicht immer der richtigen „Seite“ zugewiesen, d.h. Änderungen seitens der Entwicklung werden der Modellierung zugewiesen. Jede Änderung kann einzeln oder gesammelt angenommen oder abgelehnt werden. Nach der Bestätigung der Änderungen kam es im Laufe dieser Arbeit mehrmals vor, dass vorher generierter Code nicht den Änderungen entsprechend neu erzeugt wurde. Damit sind manuelle Anpassungen nach der Synchronisierung notwendig.

Die Workflows bilden nur eine Ebene der Anwendung, die von der Benutzeroberfläche verwendet wird und auf die Service Ebene zugreift. Diese Ebenen werden im nächsten Abschnitt beschrieben.

4.3. Implementierung und Integration

4.3.1. Implementierung der Web Services

Die Anwendungslogik verteilt sich auf die Workflow und die Web Service Ebene. Genauer gesagt definieren die Workflows den Ablauf, unter dem die Web Services aufgerufen werden (die so genannte Orchestrierung). Die Verarbeitung der Daten findet innerhalb der Web Services statt. Wie in Abschnitt 2.1.3 beschrieben, bietet ein Web Service Operationen an, die über eine Schnittstelle aufgerufen werden. Beim Import der Modellierung aus dem WBM werden die WSDL-Schnittstellen automatisch erzeugt. Auch die für die Ausführung benötigten Zusatzklassen und Einstellungen werden vom WID generiert. Alle im Fokus dieser Arbeit entwickelten Web Services werden in Java entwickelt, und innerhalb der SCA als Komponente angesehen. Dieser Abschnitt beschreibt exemplarisch den Quellcode zur Erstellung einer Auswahl der Dienstleistungen (engl.: find available services) aus Abschnitt 3.5.2.

Die Ein- und Ausgabedaten werden von der Umgebung in Form von Service Data Objects (SDOs, vgl. Abschnitt 2.2.4) von der Ausführungsumgebung verwaltet. Zur Erstellung dieser SDOs werden vorgefertigte Klassen verwendet, deren Initialisierung wie folgt implementiert wird:

```
package processes.variabilities.select_services_variability;

import com.ibm.websphere.bo.BOCopy;
import com.ibm.websphere.bo.BOFactory;
import com.ibm.websphere.sca.ServiceManager;
import commonj.sdo.DataObject;
[...]

public DataObject InputCriterion(DataObject orderBO) {
    // Get service manager singleton
    ServiceManager smgr = ServiceManager.INSTANCE;
    // Access BOFactory service
    BOFactory bof = (BOFactory) smgr
            .locateService("com/ibm/websphere/bo/BOFactory");
    // Access BOCopy Service
    BOCopy copyService = (BOCopy) ServiceManager.INSTANCE
            .locateService("com/ibm/websphere/bo/BOCopy");
```

Die `BOFactory` [IBM10] ist ein generischer Web Service, der zum Erstellen von SDOs verwendet und mit dem `ServiceManager` gefunden werden kann. SDOs werden immer baumartig miteinander verbunden, damit sie mit XML Strukturen serialisierbar sind. Das hat zur Folge, dass auf ein und dasselbe SDO immer nur einmal verwiesen werden kann. Mehrere Verweise auf dasselbe SDO werden durch Kopieren mithilfe des `BOCopy` Web Services und die Vergabe einer eindeutigen Identifikation erreicht. So wird es möglich, dass mehrere Dienstleistungen demselben Händler, d.h. mehreren SDOs mit derselben Händler ID zugewiesen werden. Als Eingabeparameter der Operation wird bereits ein SDO für ein Geschäftsdokument (`orderBO`) übergeben. Auf dieses kann direkt zugegriffen werden:

```
// get Input Values
DataObject dealer = orderBO.getDataObject("dealer");
int dealerID = dealer.getInt("ID");
String vehicleState = orderBO.getString("vehicle/state");

// Create return BO List
DataObject serviceListBO = bof.create(
        "http://eCCo5/Businessitems",
        "ListOf1ToUnboundedservices");

// Access JDBC Adapter
[...]
```

Durch den Aufruf der `BOFactory` wird ein neues SDO für die Ausgabedaten erstellt, das eine Liste aus einem oder mehreren Dienstleistungen darstellt. Alle für die Erstellung der Liste nötigen Daten sind dem Eingabe-SDO zu entnehmen. Es verweist auf das Händler SDO, von dem die ID benötigt wird, um die von ihm angebotenen Services herauszufiltern. Auf die Elemente eines SDOs wird über einen Ausdruck zugegriffen, der den Pfad im SDO-Baum festlegt. Der Ausdruck `"vehicle/state"` liefert also den Status des Fahrzeugs zurück, d.h. ob es sich um ein gebrauchtes oder ein neues Fahrzeug handelt. Dieser Status wird im nächsten Abschnitt verwendet, der die Anbindung an die Datenbank beschreibt.

4.3.2. Anbindung der Datenbank

Für die Anbindung an die Datenbank bietet der WID so genannte WebSphere Adapters [IBM11, IBM13], die vorgefertigte Module zum Zugriff auf externe Systeme darstellen. Mithilfe dieser Adapter kann beispielsweise auf Dateien oder, wie in diesem Fall, auf eine Datenbank zugegriffen werden. Der Standard zum Zugriff auf eine Datenbank im Java-Umfeld heißt Java Database Connectivity (JDBC) [SUN1]. Im Fokus dieser Arbeit wird der WebSphere Adapter für JDBC verwendet. Zunächst müssen die entsprechenden Datenbanktabellen erstellt werden. Aus dem ER-Modell aus Abschnitt 3.3 wird dazu mit JUDE/Professional [CHA1] ein Script erzeugt, mit dem die Tabellen und Felder mit den zugehörigen Eigenschaften und Schlüsseln anlegt werden. Im WID werden dann anhand eines Wizards aus den Datenbanktabellen die zugehörigen Datenobjekte und Abfragen erstellt. Der zweite Teil des Quellcodes zur Erstellung einer Auswahl der Dienstleistungen beschreibt im Folgenden, wie der Zugriff auf eine Datenbank aus einem Web Service heraus implementiert ist.

```
// Access JDBC Adapter
JDBCOutboundInterface joi =
        locateService_JDBCOutboundInterfacePartner();

// Define DB request values
DataObject serviceDO = bof.create(
        "http://eCCo5_impl/Dataitems/db2adminservice",
        "Db2adminService");

// Create DB response Object
DataObject serviceListDO = bof.create(
        "http://eCCo5_impl/Dataitems/db2adminservicecontainer",
        "Db2adminServiceContainer");

// Execute DB request
serviceListDO = joi.retrieveallDb2adminService(serviceDO);
```

Das `JDBCOutboundInterface` ist eine automatisch generierte Schnittstelle, die alle Methoden anbietet, um auf die Datenbank zuzugreifen. Jede in Java entwickelte Komponente, die auf die Datenbank zugreifen soll, muss im SCA-Diagramm mit der Komponente dieser Schnittstelle verbunden werden. Damit werden die Methoden aus dem Web Service heraus aufrufbar. Um die entsprechenden Werte abzufragen, werden zunächst zwei Datenobjekte erzeugt. Das erste dient zur Einschränkung der Abfrage, beispielsweise wenn die Ergebnismenge eingeschränkt werden soll. Das zweite Datenobjekt definiert den Ergebnistyp, in diesem Beispiel eine Liste von Dienstleistungen. Mit der Funktion `retrieveallDb2adminService` aus dem `JDBCOutboundInterface` wird die eigentliche Datenbankabfrage gestellt. Sie liefert eine Liste aller Dienstleistungen zurück, die die Eigenschaften des Beispielobjektes `serviceDO` erfüllen, d.h. alle Dienstleistungen. Sollte die Anwendung produktiv eingesetzt werden, was bei dem Prototyp im Fokus dieser Arbeit nicht der Fall ist, muss an der Stelle der Datenbankabfrage noch eine Fehlerbehandlung erfolgen.

Datenobjekte sind nicht identisch mit Geschäftsobjekten, da in Geschäftsanwendungen Daten häufig über mehrere Datenbanken und Anwendungen verteilt sind. Da der Web Service aber ein Geschäftsobjekt zurückgeben soll, müssen die Elemente des Datenobjekts noch auf die Elemente des Geschäftsobjekts abgebildet (engl.: data mapping) werden. Der zugehörige Quellcode ist im Folgenden beschrieben.

```
    // Data Mapping from DO List to BO List
    ArrayList<DataObject> serviceList = new ArrayList<DataObject>();
    List list = serviceListDO.getList("Db2adminService");
    ListIterator listIterator = list.listIterator();
    while (listIterator.hasNext()) {
        DataObject nextElement = (DataObject) listIterator.next();

        int tmpDealerID = nextElement.getInt("dealerid");
        // only add data, if dealer is unset OR current dealer
        // AND warranty extension MUST NOT be combined with a used
        // vehicle
        if (((tmpDealerID == 0) || (tmpDealerID == dealerID))
                && (!(vehicleState.equals("used") && nextElement
                        .getString("description")
                        .equals("warranty extension")))) {
            // Data mapping
            DataObject tmpServiceBO = bof.create(
                    "http://eCCo5/Businessitems", "service");
            tmpServiceBO.setInt("serviceID", nextElement
                    .getInt("serviceid"));
            if (tmpDealerID == dealerID) {
                tmpServiceBO.set("dealer", copyService.copy(dealer));
            }
            tmpServiceBO.setString("type", nextElement
                    .getString("type"));
            tmpServiceBO.setString("description", nextElement
                    .getString("description"));
            tmpServiceBO.setDouble("price", nextElement
                    .getDouble("price"));

            // add service to BO List
            serviceList.add(tmpServiceBO);
        }
    }

    // pass ArrayList to Customer List DO
    if (!serviceListBO.isSet("serviceItem")) {
        serviceListBO.setList("serviceItem", serviceList);
    }
    return serviceListBO;
}
```

Um die Werte zwischenzuspeichern, wird eine Liste namens `serviceList` angelegt. Die Regeln für die Kombination der Dienstleistungen sind der Einfachheit halber direkt im Quellcode implementiert. In anderen variablen Web Services beispielsweise der Preisberechnung, werden diese Regeln aus der `config` Datenbanktabelle gelesen. Ist dem Datenbankeintrag der Dienstleistung kein oder der aktuelle Händler zugeordnet, so wird der Eintrag der Liste hinzugefügt und alle Felder aufeinander abgebildet. Wie in Abschnitt 4.3.1 beschrieben, muss ein Datenobjekt, auf das mehrfach verwiesen wird, kopiert werden. Deshalb wird mithilfe des Befehls `copyService.copy(dealer)` eine Kopie des Händlerobjekts angelegt. Abschließend wird die zwischengespeicherte Liste dem Rückgabeobjekt übergeben und die Operation abgeschlossen.

Das Data Mapping kann auch durch so genannte „Interface Maps“ im WID grafisch modelliert werden. Dabei werden die entsprechenden Felder grafisch miteinander verbunden. Auch können mithilfe von XML Transformationen Felder zusammengefasst oder zerlegt werden. Die programmatische Abbildung wurde im Fokus dieser Arbeit aus zeitlichen Gründen bevorzugt. Sind Workflow-, Service-, und Datenebene entwickelt, fehlt der Anwendung nur noch die Präsentationsebene, mit der sich der folgende Abschnitt befasst.

4.3.3. Anbindung der Benutzeroberfläche

Um eine Benutzeroberfläche an die Anwendung anzubinden, existieren verschiedene Möglichkeiten. Durch die Human Tasks im Prozess sind alle Teile der Anwendung definiert, die eine Interaktion mit dem Benutzer benötigen. Der WID bietet drei Möglichkeiten, eine Benutzeroberfläche zu erstellen. Die erste Möglichkeit ist die Verwendung zusätzlicher Software, wie etwa Lotus Forms oder WebSphere Portal [IBM15]. Zu jedem Task kann dann im WID beispielsweise die zugehörige Portalseite angegeben werden. Vorteilhaft ist dieses Vorgehen vor allem in Unternehmen, die bereits andere Portalanwendungen im Einsatz haben und auf eine einheitliche Benutzeroberfläche wert legen. Da der Entwicklungsaufwand jedoch mit jeder Entwicklungsanwendung, steigt, wird diese Möglichkeit im Fokus dieser Arbeit nicht näher betrachtet.

Die zweite Möglichkeit ist die automatisierte Erzeugung einer Benutzeroberfläche, was gerade in den frühen Iterationen der Anwendungsentwicklung sinnvoll ist. Zu jeder Task in den modellierten Prozessen wird eine Benutzeroberfläche erzeugt, in der der Benutzer die Ein- und Ausgabedaten sehen bzw. eingeben kann. Der WID unterstützt die Erzeugung von Java Server Pages (JSPs) oder so genannten Business Spaces. Bei Letzteren kann der Anwender seine Oberfläche selbst anpassen. In beiden Fällen wird aus einer allgemeinen Vorlage eine konkret auf den Task zugeschnittene Oberfläche generiert. Abbildung 31 zeigt ein Beispiel für eine generierte Benutzeroberfläche.

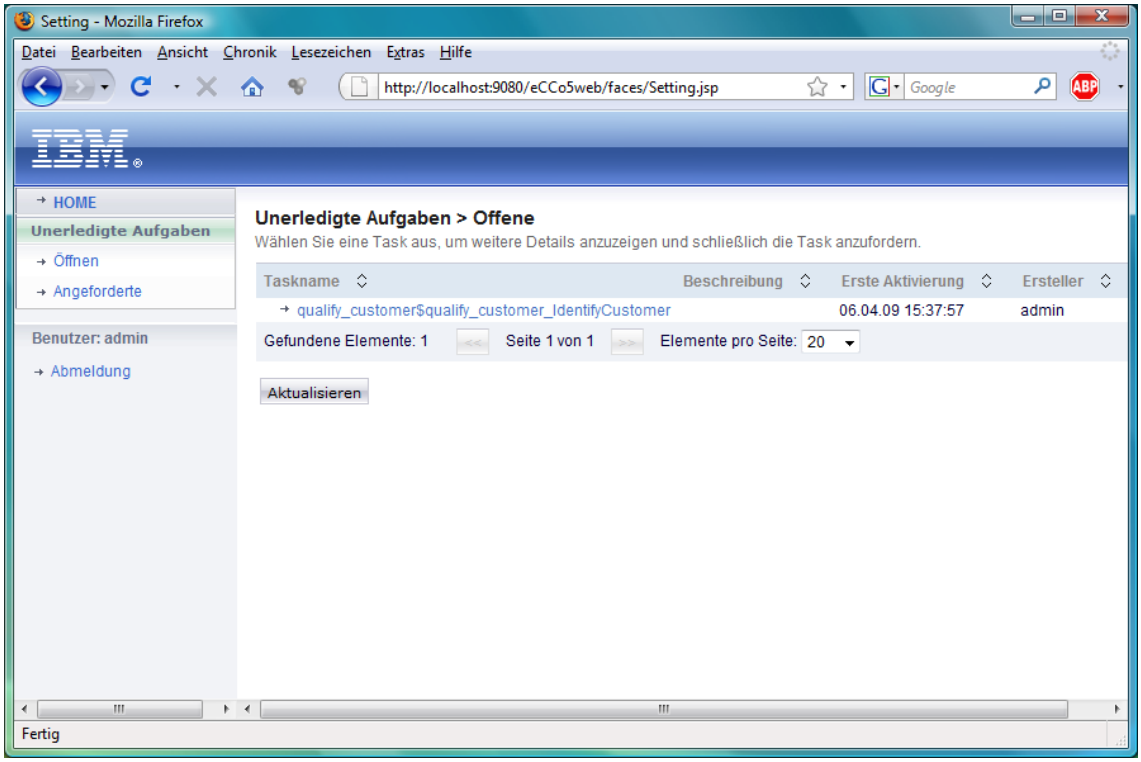

Abbildung 31: generierte Oberfläche

Für das schnelle Erstellen eines Prototyps einer Workflow-zentrierten Anwendung sind diese generierten Oberflächen durchaus geeignet. Auf diese Weise können schnell Ergebnisse präsentiert und Anforderungen besser geklärt werden. Ein wichtiger Aspekt dabei ist, dass die Anwendung immer den Workflow in den Mittelpunkt stellt. Das ist ganz im Sinne der Geschäftsautomatisierung zu sehen, bei der der Benutzer eine Liste von Aufgaben erhält, die er ähnlich zu einem Postfach abarbeiten muss. Eine benutzerzentrierte Anwendung, in der der Anwender durch eine Reihe von Seiten navigiert, ist mit diesem Modell nicht vereinbar. Da die Vorlage zur Oberflächenerstellung sehr allgemein gehalten sein muss, sind diese generierten Oberflächen häufig auch nur in geringem Maße anpassbar [IBM12]. Es kann daher durchaus sinnvoll sein, wenn die Oberfläche selbst erstellt wird.

Eine solche selbst erstellte Oberfläche ermöglicht die Nutzung eines so genannten Single Person Workflows [IBM13], bei dem der Benutzer einem festgelegten Navigationspfad folgen kann. Durch die entsprechende Auswahl kann der Benutzer neue Workflows starten und dessen Abläufen folgen. Der Angebotsprozess ist ein Beispiel dafür, da dieser zum überwiegenden Teil von einem einzelnen Händler abgearbeitet wird. Vorteilhaft ist auch, wenn der Händler das Gefühl bekommt, den Ablauf selbst zu bestimmen und nicht im Flair eines Fließbandarbeiters ein Todo-Task nach dem anderen abarbeiten muss.

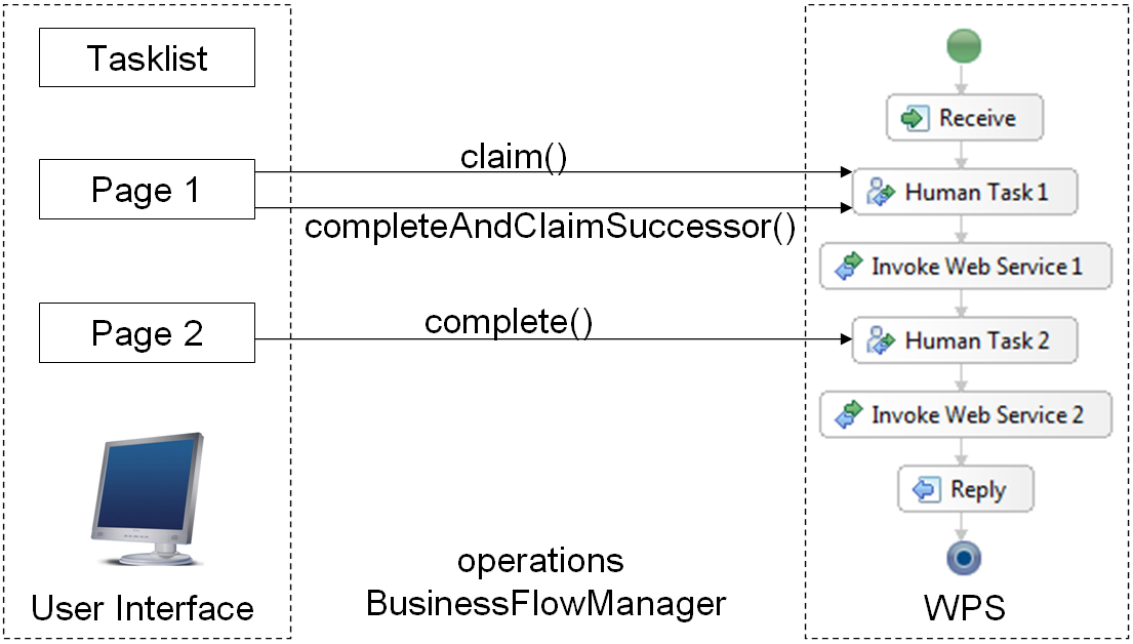

Abbildung 32: Single Person Workflow Konzept [IBM13]

Der WPS bietet eine Programmierschnittstelle an, die der selbst erstellten Oberfläche den Zugriff auf die Workflows und Human Tasks ermöglicht. Das Konzept dieser Schnittstelle ist in Abbildung 32 dargestellt. Sie ermöglicht das Abfragen aktuell zu bearbeitender Tasks oder laufender Workflowinstanzen. Um ein Angebot zu erstellen, startet der Benutzer den Angebotsprozess. Die entsprechenden Unterprozesse und Tasks werden automatisch vom WPS gestartet. Vom Programmcode der Oberfläche muss dann periodisch nach zu bearbeitenden Tasks gesucht werden. Wird ein solcher gefunden, wird er automatisch zur Bearbeitung angenommen und der Benutzer auf die nächste Seite weitergeleitet. Das hat zur Folge, dass die Benutzeroberfläche bei jeder Prozessänderung angepasst werden muss.

Die Wiederverwendung von Tasks, beispielsweise das Anzeigen einer Bestätigung ist im Kontext einer selbst erstellten Benutzeroberfläche nicht empfehlenswert, da globale Tasks keinen Kontext in Bezug auf den übergeordneten Workflow besitzen. Da die Logik der Anwendung und der Fluss der Benutzerseiten nicht zusammenpassen, ist die Implementierung eines Single Person Workflows eine Gratwanderung, bei der einige Abstriche bei der Benutzerfreundlichkeit gemacht werden müssen. In der im Fokus dieser Arbeit verwendeten Version des WPS ist es beispielsweise noch nicht möglich einzelne Tasks zu überspringen oder erneut zu durchlaufen.

Die in Abbildung 33 beispielhaft abgebildete Oberfläche wurde aus Zeitgründen nicht vom Autor dieser Arbeit erstellt, und ist daher im Folgenden nicht detailliert beschrieben. Da alle Geschäftsprozesse im Fokus dieser Arbeit mit Ausnahme der Bestätigung von Bestellungen von Anwendern derselben Rolle ausgeführt werden, wird eine Lösung mit selbst erstellten JSPs favorisiert.

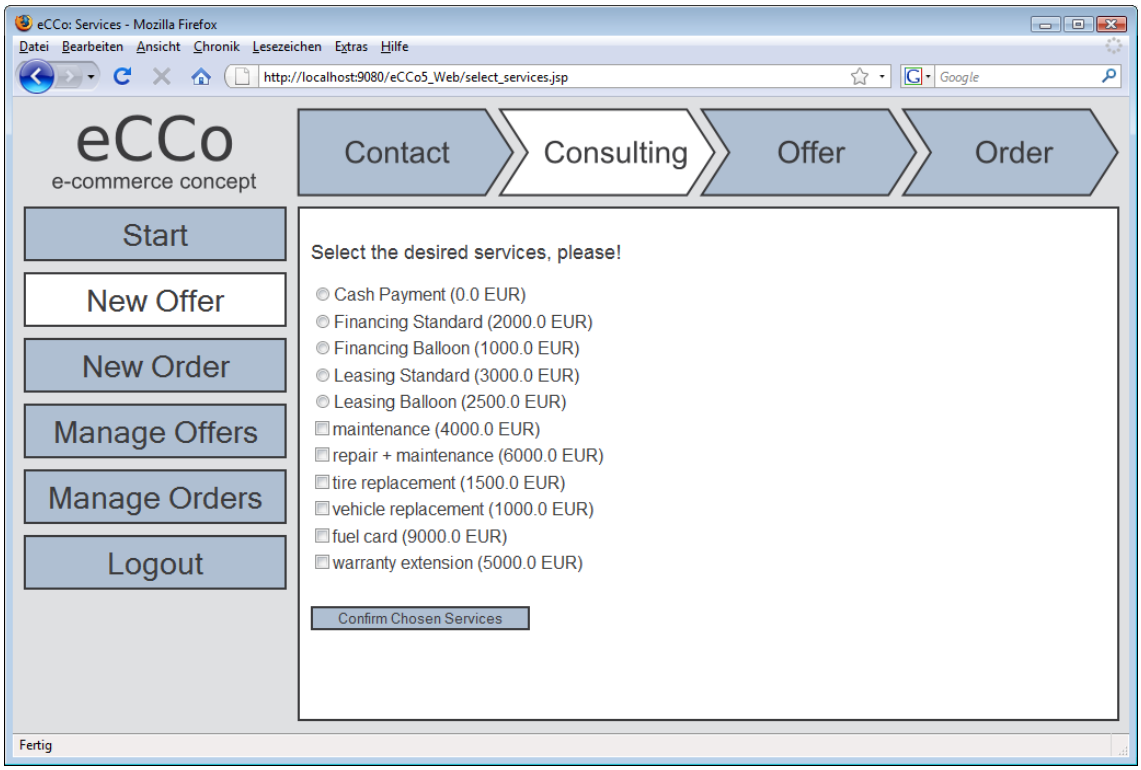

Abbildung 33: Single Person Workflow Oberfläche

Auf der linken Seite von Abbildung 33 ist ein Menü zu sehen, mit dem die verschiedenen Workflows gestartet werden können. Am oberen Rand wird die aktuelle Position in der Wertschöpfungskette angezeigt. Im Arbeitsbereich wird eine Liste von verfügbaren Dienstleistungen angezeigt, aus denen der Benutzer auswählen kann. Mit Klick auf die Schaltfläche zum Bestätigen wird er auf die nächste Seite weitergeleitet.

Die letzte und einfachste Möglichkeit eine Benutzeroberfläche zu nutzen, stellt der Business Process Choreographer Explorer (BPCE) dar. Innerhalb des im Fokus dieser Arbeit angewendeten Entwicklungsprozesses wird die Benutzeroberfläche erst für den Test des kompletten Systems benötigt wird. Um Wiederholungen zu vermeiden, wird der BPCE in Kapitel 5 beschrieben. Der folgende Abschnitt beschreibt einen Überblick über die einzelnen Module der Anwendung.

4.3.4. Modulübersicht

Nach der Erstellung der Benutzeroberfläche ist die Anwendung fertiggestellt und kann auf einem WPS installiert werden. Abbildung 34 gibt zusammenfassend eine Übersicht über die Module der Anwendung und zeigt deren Zuordnung zu den Ebenen der Architektur.

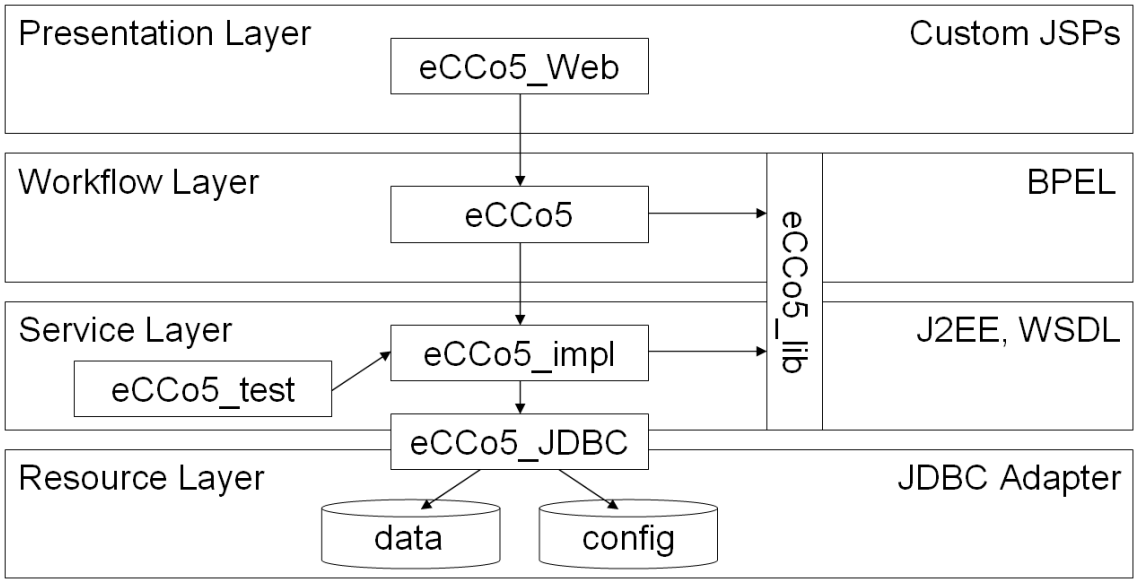

Abbildung 34: Modulübersicht

Das Modul `eCCo5_Web` enthält die eigens für die Anwendung erstellten JSPs, durch die der Benutzer Zugriff auf die Anwendung erhält. Das Modul `eCCo5` enthält alle Workflows, deren Import in Abschnitt 4.2.2 beschrieben ist. Die in Abschnitt 4.3.1 beschriebene Implementierung der Web Services befindet sich im Modul `eCCo5_impl`. Beide Module greifen auf die gemeinsame Bibliothek `eCCo5_lib` zu, die die Datentypen und Schnittstellen enthält. Der in Abschnitt 4.3.2 beschriebene JDBC Adapter ist im Modul `eCCo5_JDBC` zu finden. Das Modul `eCCo5_test` enthält eine Testsuite für das Testen der Web Services, das im nächsten Kapitel beschrieben ist.

5. Test der Anwendung

5.1. Testszenarien

Gerade bei der iterativen Entwicklung einer Anwendung ist eine geeignete Testumgebung erforderlich. Der Test der Anwendung gliedert sich in drei Bereiche, die an den in Abschnitt 4.1.2 beschriebenen Entwicklungsprozess angelehnt sind. Der erste Bereich ist der Test der Workflows. Workflows isoliert vom Rest der Anwendung zu testen ist nur bedingt möglich, da der Kontrollfluss der Workflows teilweise von den Rückgabewerten der einzelnen Aktivitäten abhängt. Beim Test der Web Services werden diese mit dem zugrundeliegenden Quellcode zunächst isoliert voneinander getestet. Aus diesen isolierten Tests wird ein Regressionstest erstellt, der bei Änderungen beispielsweise des Datenmodells die Tests der einzelnen Web Services gesammelt ausführt. Sind die Workflows und die Web Services lauffähig, kann in einem Systemtest die Einhaltung der Anforderungen anhand vorher festgelegter Testfälle evaluiert werden. Parallel zur Entwicklung der Benutzeroberfläche kann dann die Anbindung der Datenbank entwickelt werden. Diese drei Bereiche sind im folgenden Abschnitt beschrieben.

5.2. Durchführung der Tests

5.2.1. Test der Worflows

Die modellierten Workflows können direkt nach dem Import auf dem WPS installiert und ausgeführt werden. Dazu wird eine Oberfläche zum Starten und Bearbeiten der Workflows benötigt. Im WPS ist eine Webanwendung namens Business Process Choreographer Explorer (BPCE) [IBM14] enthalten, die dies ermöglicht. Mit diesem kann eine Übersicht aller Workflow Templates angezeigt werden. Abbildung 35 zeigt die Ansicht der vom Anwender verwalteten Workflowinstanzen im BPCE.

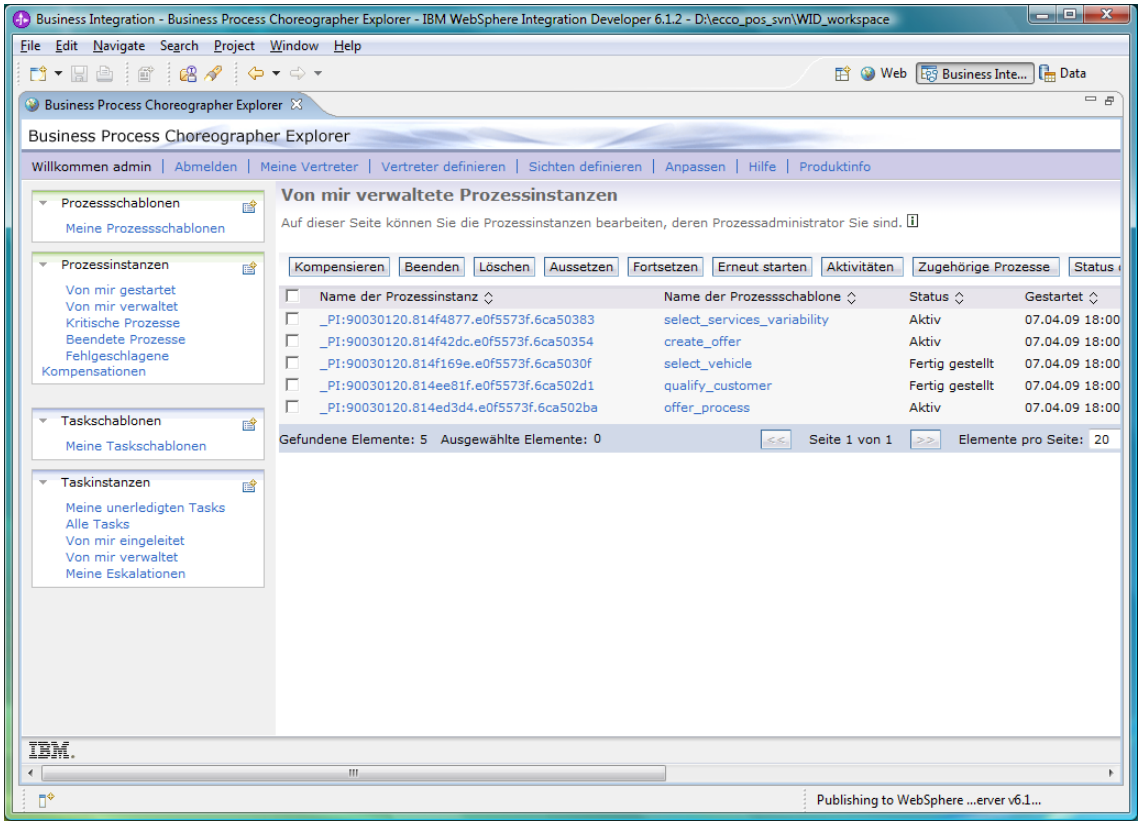

Abbildung 35: BPCE - Verwaltung von Prozessinstanzen

Des Weiteren ist es möglich, direkt aus dem BPCE Prozessinstanzen mit den gewünschten Eingabewerten zu starten und zu beenden bzw. zu löschen. Für das Testen der Workflows können alle relevanten Details, wie beispielsweise Workflowinstanz IDs, Eingabe- und Ausgabedaten oder aufgetretene Fehler angezeigt werden. Im Rahmen eines Tests wird ein Prozess der obersten Ebene gestartet. Die Unterprozesse starten dann automatisch. Alle Eingaben, die später vom Benutzer getroffen werden, sind einzugeben. Dabei empfiehlt es sich, das automatische Löschen abgeschlossener Workflows zu deaktivieren, um die Ausgabenachricht der einzelnen Workflows zu prüfen.

Die von Web Services ausgeführten Aktivitäten eines Workflows geben zum Test selbiger Beispieldaten zurück. Diese werden zunächst fest im Quellcode hinterlegt, bis alle Workflows lauffähig sind. Der Test der im Anschluss durchgeführten Implementierung der Web Services ist im nächsten Abschnitt beschrieben.

5.2.2. Test der Web Services

Ein Web Service stellt eine von anderen gekapselte Komponente der Anwendung dar. Diese Komponenten sind deshalb isoliert voneinander zu entwickeln und zu testen. Für das Testen einzelner Web Services wird der im WID integrierte Universal Test Client (UTC) [IBM13] verwendet, der in Abbildung 36 dargestellt ist.

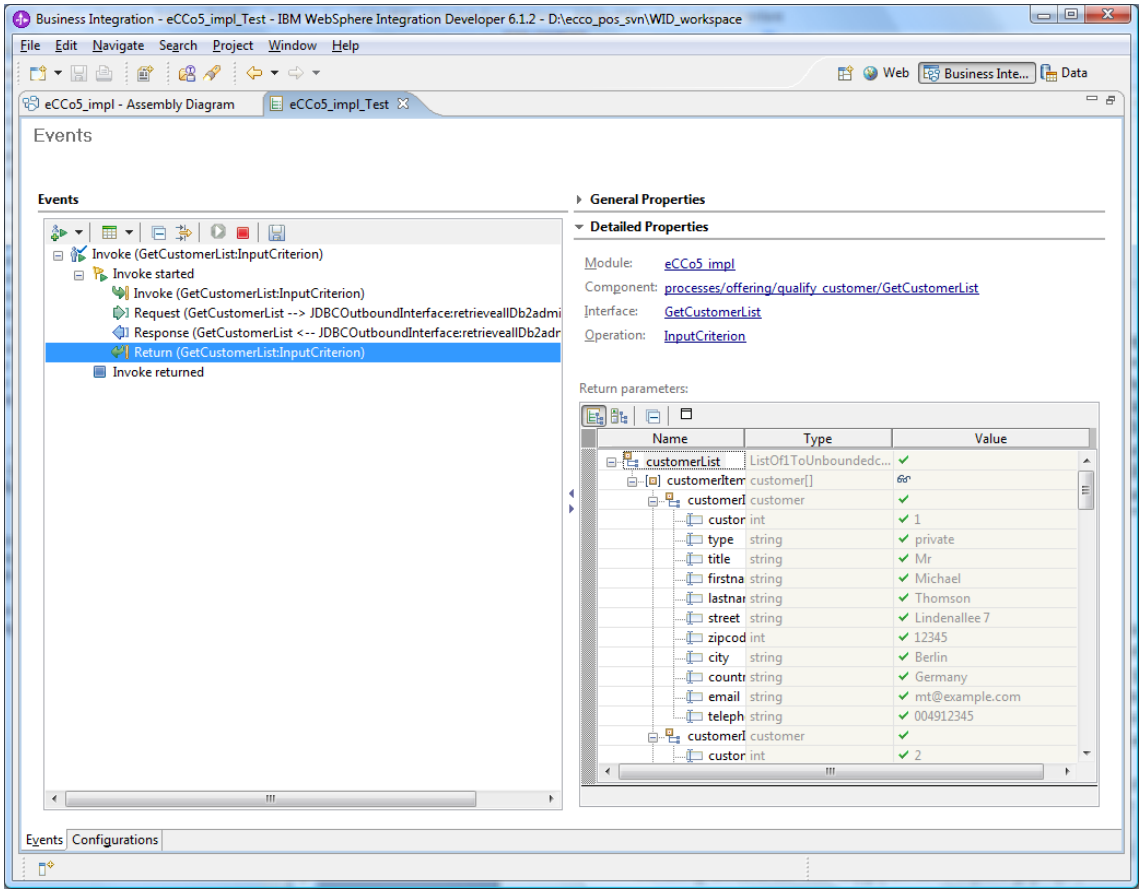

Abbildung 36: Web Service Test

Auf der linken Seite des UTC werden Testfälle angelegt. Zu jedem Testfall werden innerhalb einer `<invoke>` Operation Eingabedaten für den Aufruf eingegeben. Anschließend wird der Web Service mit diesen Eingabedaten aufgerufen. In Abbildung 36 wird ein Aufruf des Web Services zur Abfrage der Kundenliste gezeigt. Dieser Web Service fragt die Daten aus der Datenbank ab, was als request/ response Paar angezeigt wird. Alternativ kann auch ein isolierter Testmodus gewählt werden, bei dem alle Aufrufe externer Services emuliert werden. In diesem Fall können die Rückgabewerte direkt vom Tester eingegeben werden. Das ist vor allem dann sinnvoll, wenn die aufzurufenden Dienste noch nicht zur Verfügung stehen. Wie in Abschnitt 4.3.1 beschrieben, werden die erhaltenen Daten in ein Geschäftsobjekt geschrieben und zurückgegeben. Die Rückgabewerte werden in Abbildung 36 auf der rechten Seite in einer Baumstruktur angezeigt.

Für die Bewertung eines Testergebnisses wird ein Soll-Ergebnis benötigt, dass mit dem zurückgegebenen Ist-Ergebnis verglichen wird. Im Fall der Abfrage einer Kundenliste kann das Soll Ergebnis der Datenbanktabelle entnommen werden. Da ein zustandsloser Web Service zu denselben Eingabedaten immer dieselben Ausgabedaten liefert, kann ein solcher Test mehrfach wiederholt werden. Um dafür die Eingabewerte nicht erneut eingeben und die Soll- nicht manuell mit den Ist-Ergebnissen vergleichen zu müssen, können diese in Form eines Testfalls gespeichert werden. Alle diese Testfälle werden zu einem Regressionstest zusammengefasst, der im WID „Testsuite“ genannt wird. Abbildung 37 stellt die Testsuite für den Regressionstest dar.

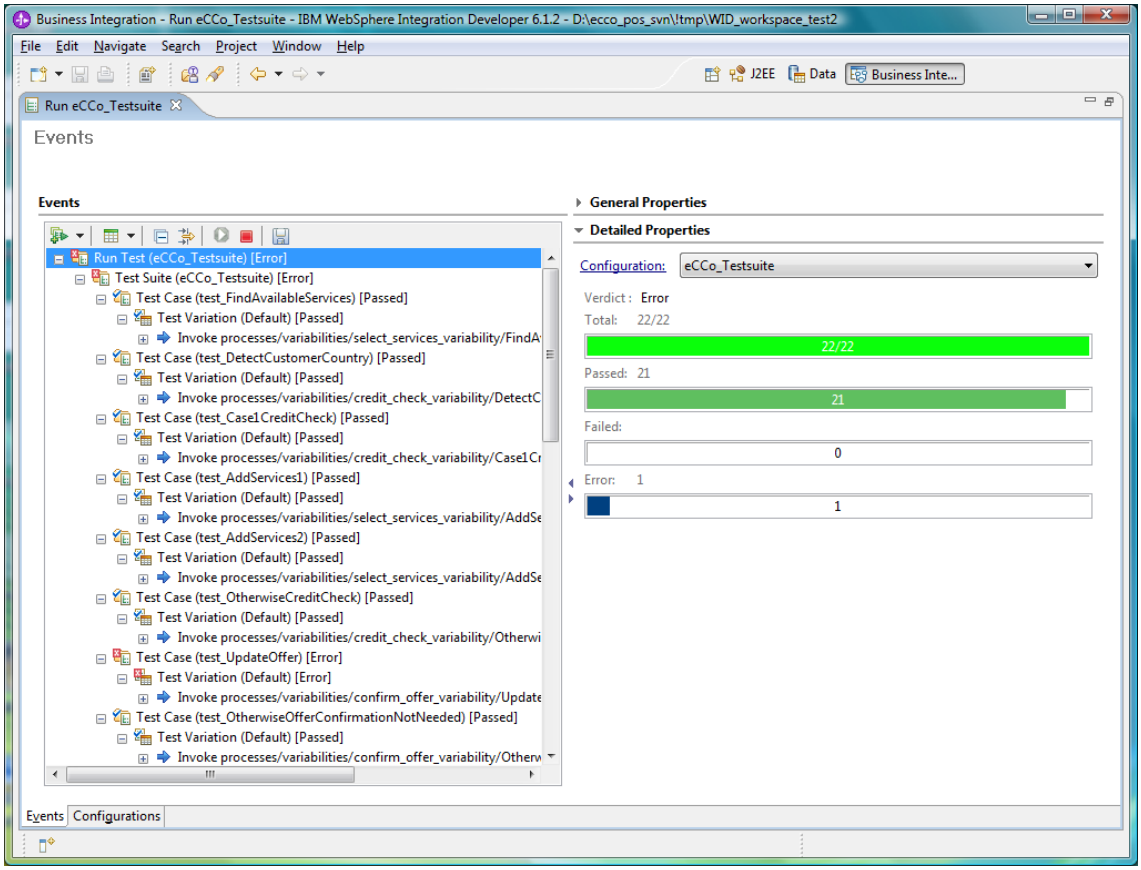

Abbildung 37: Regressionstest

Der Regressionstest definiert ähnlich zu einem klassischen Modultest drei Ergebnisse für einen Testfall. Wurde er erfolgreich ausgeführt und stimmen Soll- und Ist-Ergebnis überein, so ist das Ergebnis „erfolgreich“. Unterscheiden sich Soll- und Ist-Ergebnis, schlägt der Test fehl. Dies ist beispielsweise für das Prüfen von Fehlerbehandlungen sinnvoll. Wenn die Ausführung eines Testfalls nicht erfolgreich ist, so erhält das Ergebnis den Status „Fehler“. Alle Ergebnisse werden nach Kategorie sortiert in einer Übersicht angezeigt. Die Übersicht in Abbildung 37 zeigt 21 erfolgreiche Ergebnisse und einen Fehler. Auf diese Weise können einmal erstellte Testfälle wiederholt ausgeführt werden. Eine entsprechende Qualität der Testfälle vorausgesetzt, dient der Regressionstest der zuverlässigen Prüfung der Auswirkungen von Änderungen beispielsweise am Datenmodell.

Da nicht alle Web Services ohne Zustandsinformationen implementierbar sind, weil sie beispielsweise Daten in die Datenbank schreiben, müssen die Testfälle nach Anbindung einer Datenbank meist überarbeitet werden. Mithilfe von Datenbankskripten ist vor Ausführung eines Regressionstests ein fest definierter Zustand in der Datenbank herzustellen. Dieser Weg wurde im Fokus dieser Arbeit gewählt, um den Regressionstest weiter verwenden zu können. Nach Ausführung des Tests ist der Zustand in der Datenbank zu überprüfen, was automatisiert werden kann. Für einen Test der kompletten Anwendung reicht der Regressionstest mangels Prüfung der Benutzeroberfläche nicht aus. Dafür wird der Systemtest verwendet, der im nächsten Abschnitt beschrieben ist.

5.2.3. Systemtest

Bevor ein Systemtest durchgeführt werden kann, muss die Oberfläche erstellt werden. Da die in Abschnitt 4.3.3 beschriebenen JSPs die vom WPS zur Verfügung gestellte Task Schnittstelle intensiv nutzten, ist es überaus hilfreich, eine Oberfläche zum Anzeigen und zur Verwaltung aktuell laufender Tasks zu nutzen. Für diese Aufgabe ist der BPCE wie in Abbildung 38 dargestellt, gut geeignet.

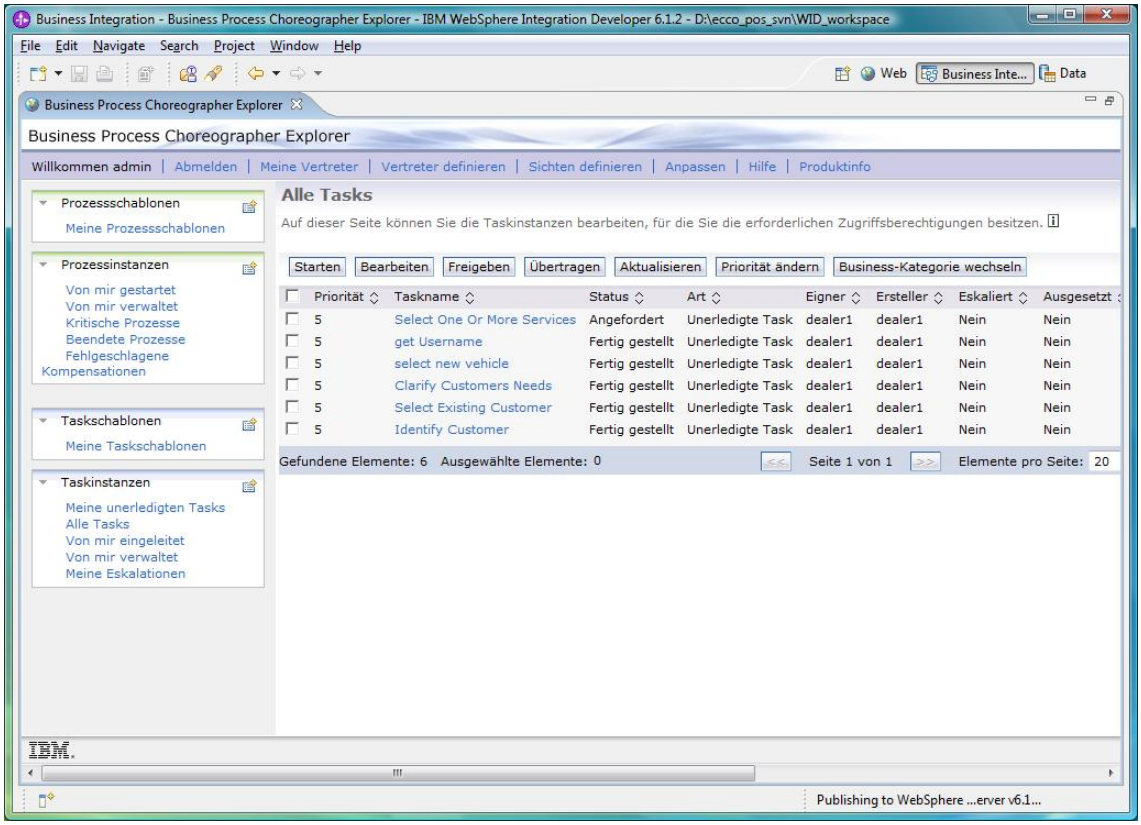

Abbildung 38: BPCE - Verwaltung von Human Tasks

Der BPCE dient damit der Ergänzung der Oberflächenentwicklung. So kann geprüft werden, ob die Tasks zur Bearbeitung angenommen werden und nach der Bearbeitung abgeschlossen werden.

Nach der Implementierung der Benutzeroberfläche wird mit deren Hilfe der Systemtest durchgeführt. Der Systemtest ist auf Grundlage der in Abschnitt 3.2.2 beschriebenen Anwendungsfälle gestaltet. Dabei werden zum regulären Ablauf, der in den Geschäftsprozessen aus Abschnitt 3.4 beschrieben ist, Szenarien angegeben. Außerdem wird zu jedem alternativen Ablauf der Anwendungsfälle mindestens ein Testfall beschrieben. Tabelle 11 zeigt einen Auszug aus dem Systemtest für den Anwendungsfall „Angebot erstellen“.

Nr.	**Bezeichnung**	
12	*Anfangszustand*	• Dialog „select services“ wird angezeigt • dealer1 (Deutschland) ist angemeldet
	Eingabe(n)	„Financing Standard“ auswählen und bestätigen
	Soll-Resultat	• vor Bestätigung keine Aftersales Services auswählbar • Dialog „show confirmation“ wird angezeigt • Gesamtpreis mit 19% MWSt berechnet
13	*Anfangszustand*	• Dialog „select services“ wird angezeigt • dealer1 (Deutschland) ist angemeldet
	Eingabe(n)	„Leasing Balloon“ auswählen und bestätigen
	Soll-Resultat	• vor Bestätigung weitere Aftersales Services von dealer1 mit dessen Preisen auswählbar • Dialog „show confirmation“ wird angezeigt • Gesamtpreis mit 19% MWSt berechnet
14	*Anfangszustand*	• Dialog „select services“ wird angezeigt • dealer1 (Deutschland) ist angemeldet
	Eingabe(n)	„Leasing Standard“ und „maintenance“ auswählen und bestätigen
	Soll-Resultat	• vor Bestätigung weitere Aftersales Services von dealer1 mit dessen Preisen auswählbar • Dialog „confirm offer“ wird angezeigt • Gesamtpreis mit 19% MWSt berechnet
15	*Anfangszustand*	• Dialog „select services“ wird angezeigt • dealer2 (Spanien) ist angemeldet
	Eingabe(n)	„Financing Balloon“ auswählen und bestätigen
	Soll-Resultat	• vor Bestätigung keine Aftersales Services auswählbar • Dialog „confirm offer“ wird angezeigt • Gesamtpreis mit 16% MWSt (Spanien) berechnet

16	*Anfangszustand*	• Dialog „select services“ wird angezeigt • dealer2 (Spanien) ist angemeldet
	Eingabe(n)	„Cash Payment“, „tire replacement“ und „fuel card“ auswählen und bestätigen
	Soll-Resultat	• vor Bestätigung weitere Aftersales Services von dealer2 (ohne „maintenance“) mit dessen Preisen auswählbar • Dialog „confirm offer“ wird angezeigt • Gesamtpreis mit 16% MWSt (Spanien) berechnet
17	*Anfangszustand*	• Dialog „select services“ wird angezeigt • dealer3 (Deutschland) ist angemeldet • für dealer3 sind keine Aftersales Services gepflegt
	Eingabe(n)	„Cash Payment“ auswählen und bestätigen
	Soll-Resultat	• vor Bestätigung sind keine Aftersales Services auswählbar • Dialog „show confirmation“ wird angezeigt • Gesamtpreis mit 19% MWSt berechnet
18	*Anfangszustand*	• Dialog „select services“ wird angezeigt • dealer1 (Germany) ist angemeldet • Gebrauchfahrzeug wurde ausgewählt
	Eingabe(n)	„Cash Payment“ auswählen
	Soll-Resultat	• Der Aftersales Service „warranty extension“ ist nicht auswählbar
19	*Anfangszustand*	• Dialog "select services" wird angezeigt • dealer1 (Germany) ist angemeldet
	Eingabe(n)	keine Auswahl treffen und bestätigen
	Soll-Resultat	• Fehlermeldung „You have to select at least one Financial Service.” wird angezeigt
20	*Anfangszustand*	• Dialog „select services“ wird angezeigt • dealer1 (Germany) ist angemeldet
	Eingabe(n)	„fuel card“ auswählen und bestätigen
	Soll-Resultat	• Fehlermeldung „You have to select at least one Financial Service.” wird angezeigt

Tabelle 11: Systemtest Beispiele

Bei der Durchführung des Systemtests werden die Ergebnisse und eventuelle Bemerkungen in einem Protokoll dokumentiert. Auch das Verhalten bei Fehleingaben und fehlerhaften Daten sind vom Systemtest zu berücksichtigen. Er dient damit der Evaluierung, ob die Anforderungen, die an die Anwendung gestellt wurden, erfüllt werden. Die im nächsten Abschnitt beschriebenen Erkenntnisse gehen aus dem Ausführen des Systemtests hervor.

5.3. Auswertung

Schon bei der Entwicklung der Benutzeroberfläche fallen Fehler in der Modellierung auf. Beispielsweise wurde die Aktivität getUsername aus Abbildung 17 nachträglich hinzugefügt, damit der für den Kontext wichtige Benutzername dem Workflow übergeben wird. Aus der Ausführung des Systemtests können oft Empfehlungen für die Verbesserung der Benutzerfreundlichkeit abgeleitet werden. So erscheint es beispielsweise wünschenswert, wenn beim Ändern eines Angebots, die vorher ausgewählten Dienstleistungen angezeigt werden. An dieser Stelle wird jedoch der Workflow der Angebotserstellung wiederverwendet, so dass eine Änderung des Datenflusses auf mehrere Geschäftsprozesse Auswirkungen hat. Auch erzeugt jede Änderung einer Bestellung einen Task zur Prüfung derselben, was bei häufig durchgeführten Änderungen wenig sinnvoll erscheint. Wie in Abschnitt 4.3.3 beschrieben, führt auch die Wiederverwendung von Human Tasks in mehreren Workflows zu einer Reihe technischer Probleme.

Im Rahmen der Beispiele dieser Arbeit entsteht in der Tat kein signifikanter Mehrwert aus der Wiederverwendung ganzer Workflows. Der niedrigeren Zahl an Prozessmodellen steht die fehlende Flexibilität und höhere Komplexität bei Änderungen gegenüber. Parametrisierte Workflows erhöhen die Komplexität ebenfalls. Dass der Kontext, in dem ein Prozess aufgerufen wird, kaum an anderer Stelle gleich sein wird, ja sogar von einem ganz anderen Bereich entwickelt werden kann, bekräftigt diese These. Lohnenswert hingegen ist die Wiederverwendung einzelner Web Services, da auf diese Weise Mehrfachimplementierungen vermieden werden können. Da ein Web Service weniger komplex als die übergeordneten Workflows ist, überwiegen bei diesen die Vorteile einer Wiederverwendung. Im Rahmen der Implementierung ist eine Parametrisierung leicht und ohne größere Performanzeinbußen umsetzbar. Wird die Konfiguration einzelner Werte noch ausgelagert, entstehen eine Reihe von Vorteilen, die im nächsten Kapitel beschrieben sind.

6. Evaluierung der Ergebnisse

6.1. Deployment Strategien

6.1.1. Vorbemerkungen

Durch die Globalisierung gehört das Erschließen neuer Märkte für internationale Automobilkonzerne zur Realität. Auch ein Vertriebsnetz muss hinreichend flexibel erweiterbar sein. Wünschenswert ist außerdem, dass Geschäftsmodelle zügig an sich verändernde Wettbewerbssituationen angepasst werden können. Daraus werden drei wichtige Szenarien abgeleitet: Das Hinzufügen neuer Märkte und Händler sowie die Veränderung einer Berechnung bzw. eines Ablaufs. Da innerhalb eines Marktes viele Händler existieren, sind das Hinzufügen und Löschen von Händlern die am häufigsten durchgeführten Szenarien. Soll eine gemeinsame Anwendung verwendet werden, so sind deren Ausfallzeiten im Hinblick auf die ständige Nutzung, die durch die Zeitverschiebung entsteht, möglichst minimal zu halten.

Für das Deployment der geänderten Anwendung kommen verschiedene Ansätze in Frage. Werden alle Workflows einer Anwendung neu installiert, muss mit Störungen im Betrieb gerechnet werden. Enthalten die Workflows variable Unterprozesse, so verlagert sich das Problem zwar eine Ebene tiefer, bleibt aber bestehen. Web Services können mithilfe eines ESB dynamisch an die Anwendung gebunden werden, d.h. die Referenz auf den Web Service wird erst zur Laufzeit festgelegt. Damit können Web Services zur Laufzeit ausgetauscht werden, ohne den Betrieb zu stören. Ändert sich jedoch ein Ablauf, d.h. der Kontrollfluss in einem Workflow, reicht das dynamische Binden eines anderen Web Services nicht aus, um die Änderung abzubilden. Das hat zur Folge, dass das erneute Deployment von Workflows mitbetrachtet werden muss.

Wird der WebSphere Business Modeler Advanced für die Modellierung der Geschäftsprozesse verwendet, so müssen wie in Abschnitt 4.2.1 beschrieben die Geschäftsprozesse global modelliert werden. Dies gilt insbesondere für die Variabilitäten, da diese die sich häufig ändernden Teile der Geschäftsprozesse abbilden. Durch die Versionierung von Prozessmodellen kann eine Variabilität separat in einer neuen Version installiert werden, ohne dass der übergeordnete Workflow geändert werden muss.

6.1.2. Hinzufügen von Märkten und Händlern

In der im Fokus dieser Arbeit beschriebenen Anwendung kann ein neuer Händler hinzugefügt werden, ohne dass die Anwendung neu installiert wird. Der Händler und optional die von ihm angebotenen Dienstleistungen werden in die Datenbank eingetragen und der Benutzer in der Serverkonfiguration angegeben. Das Anlegen eines neuen Händlers in einem vorhandenen Markt ist damit zur Laufzeit möglich. Die Ebene der Web Services muss nicht geändert werden, wodurch die Anwendung weiterlaufen kann.

Das Anlegen neuer Märkte ist etwas aufwändiger als das Anlegen neuer Händler. Wenn auf einen vorhandenen Web Service zugegriffen werden kann, genügt ein Eintrag für die Mehrwertsteuer in der Konfigurationstabelle. Gegebenenfalls muss aber ein neuer Web Service für die Kreditprüfung angebunden werden. Mit dem im WPS integrierten Enterprise Service Bus kann dieser Service dann innerhalb einer Variabilität dynamisch angebunden werden. Bei dem für diese Arbeit entwickelten Prototyp ist das dynamische Binden nicht implementiert. Stattdessen muss eine neue Version der in Abschnitt 3.5.2 beschrieben Variabilität „Kreditwürdigkeit prüfen“ installiert werden. Empfehlenswert ist daher die Auslagerung der variablen Workflows in ein eigenes Modul, was durch die zugrundeliegende SCA leicht umsetzbar ist.

6.1.3. Änderungen am Geschäftsmodell

Um Änderungen am Geschäftsmodell abzubilden, muss die Art der Änderung betrachtet werden. So unterscheidet sich das Vorgehen bei einer Berechnungsänderung von der Änderung des Kontrollflusses. Für eine Änderung der Preisberechnung muss die Implementierung des entsprechenden Web Services verwendet werden. Da dieser innerhalb einer Variabilität aufgerufen wird, kann im Fall des dynamischen Bindens ein neuer Web Service installiert werden, der dann durch eine neue Variabilitätsversion zur Laufzeit angebunden wird. Eine Alternative ist die Versionierung von Web Services, die im Rahmen dieser Arbeit nicht näher betrachtet wird.

Ändert sich der Kontrollfluss eines Workflows, so muss eine neue Version dieses Workflows installiert werden. Durch die Benutzung global modellierter Workflows können die übergeordneten Workflows bestehen bleiben. Kommen neue Web Service Aufrufe hinzu, so müssen diese Web Services ebenfalls installiert werden. Durch den in Abschnitt 4.1.2 beschriebenen Entwicklungsprozess wird dafür zunächst das Geschäftsprozessmodell geändert und anschließend mit der Implementierung synchronisiert. Dies ist aufwändiger, als das direkte Ändern des Workflows, ermöglicht aber eine zur Implementierung konsistente Dokumentation der Prozesse, die von den Fachbereichen verstanden wird. Ändert sich die Menge oder die Reihenfolge der verwendeten Human Tasks so muss die Benutzeroberfläche entsprechend angepasst werden.

6.2. Dienstgüteeigenschaften

Dienstgüteeigenschaften (engl.: quality of services) einer Software haben großen Einfluss auf die Gesamtkosten einer Anwendung. Nach einer gewissen Einsatzzeit machen die Wartungskosten den größten Teil der Kosten einer Software aus. Es ist deshalb sinnvoll, schon bei der Entwicklung Maßnahmen zu ergreifen, die die Dienstgüteeigenschaften einer Software erhöhen und damit die Wartungskosten senken.

6.2.1. Betrachtung der Anwendung

Dabei hängen große Teile der Dienstgüteeigenschaften, wie Skalierbarkeit, Robustheit und Portabilität von den der Anwendung zugrundeliegenden Entwicklungs- bzw. Laufzeitumgebung ab. Die im Rahmen dieser Arbeit verwendete Umgebung erleichtert die Skalierbarkeit durch Verteilbarkeit des Anwendungsservers (vgl. Abschnitt 4.1.1). Portabilität ist durch die Verwendung plattformunabhängiger Sprachen, wie BPEL, Java und WSDL zu erreichen. Robustheit hingegen ist keine Stärke des WID und der enthaltenen Komponenten. Gerade bei Änderungen an der im Fokus dieser Arbeit entwickelten Anwendung sind häufig Fehler bei der Installation oder Deinstallation aufgetreten, die häufig nach Neustart der Laufzeitumgebung nicht mehr auftraten. Auch die Bereinigungsfunktion der Entwicklungsumgebung, die die generierten Teile der Anwendung neu erzeugt, beseitigte oft Fehler. Beispielsweise werden nach dem Import des Arbeitsbereichs auf einem anderen Computer zwei Fehler im Bezug auf die JDBC Schnittstelle gemeldet. Nach dem Löschen und Benutzen der „Rückgängig“-Funktion verschwinden diese Fehler plötzlich. Zu nicht rekonstruierbaren Laufzeitfehlern kam es jedoch äußerst selten. Da die Laufzeitumgebung in vielen Bereichen im Einsatz ist, kann ein angemessenes Maß an Robustheit vorausgesetzt werden.

Die Wartbarkeit einer Anwendung ist die Dienstgüteeigenschaft mit den wohl umfangreichsten Auswirkungen auf die Wartungskosten. Wichtige Aspekte der Wartbarkeit sind Modularisierung, Verständlichkeit und Dokumentation. Diese Eigenschaften sind vor allem wichtig für zukünftige Anpassungen. Einzelne Komponenten müssen klar voneinander getrennt werden, unter anderem auch um den isolierten Test einzelner Komponenten zu ermöglichen. Dies ist durch die Verwendung einer SCA und der verwendeten Entwicklungsumgebung in hohem Maße gegeben. Workflows und Web Services sind durch definierte Schnittstellen klar voneinander getrennt und können damit leichter ausgetauscht werden. Die Verständlichkeit der Anwendung wird vor allem durch die Verwendung von bekannten Entwurfsmustern und verbessert. So ist beispielsweise die Einhaltung definierter Architekturebenen ein wichtiges Kriterium für das Verständnis komplexer Systeme. Die Strukturen, die einer Anwendung zugrunde liegen, müssen dokumentiert sein, damit sich diejenigen, die die Wartung durchführen sollen, sich leichter einarbeiten können. Dokumentation wird bei Softwareentwicklungsprojekten häufig vernachlässigt. Durch den in Abschnitt 4.1.2 beschriebenen Entwicklungsprozess stimmt die grafische Darstellung der Workflows zu jeder Zeit mit der Implementierung überein. Da die Workflows einen großen Teil der Anwendungslogik beschreiben, sind diese ein wichtiger Faktor für eine erfolgreiche Weiterentwicklung.

6.2.2. Einfluss der Variabilitäten

Variabilitäten beschreiben alle kontextabhängigen Elemente der Workflows. Damit wird an einer zentralen Stelle dokumentiert, welche Teile der Anwendung abhängig vom Benutzer sind. Dies erleichtert die Einarbeitung der Entwickler, die die Anwendung implementieren bzw. warten. Patterns (vgl. Abschnitt 2.3.3) haben sich in der Softwareentwicklung bewährt, wodurch die einheitliche Verwendung von Multi-Tenancy-Patterns die Verständlichkeit der Anwendung weiter verbessert wird.

Der größte Vorteil der Verwendung von Variabilitäten ist allerdings, dass häufig vorkommende Erweiterungsszenarien, wie etwa das Hinzufügen neuer Händler oder Märkte keine Unterbrechung des Betriebs erfordern. Meist müssen in solchen Fällen nur Konfigurationen geändert werden. Dies erweitert die flexible Nutzung parametrisierbarer Services auf die gesamte Anwendung, was im Begriff „Software as a Service“ durchaus treffend beschrieben ist. In Verbindung mit der ebenfalls gegebenen Skalierbarkeit entsteht eine Anwendung mit stark gesenkten Wartungskosten. Entsprechend qualifiziertes Entwicklungspersonal vorausgesetzt, können auch die gesteigerte Komplexität und die geringere Performanz diese Senkung nicht kompensieren.

6.3. Fazit

Durch die Implementierung der Konzepte in Form eines Prototyps ist gezeigt, dass die Konzepte der Variabilitäten und der Multi-Tenancy-Patterns umsetzbar sind. Richtig umgesetzt ist mit ihnen eine Verbesserung der Dienstgüteeigenschaften einer Anwendung erreichbar. Vor allem die Wartbarkeit und Skalierbarkeit profitieren von der Verwendung parametrisierter Systeme und Komponenten. Soll eine Anwendung mit Workflow Technologie entwickelt werden, so steigert der verwendete Entwicklungsprozess die Qualität der Dokumentation. Eingeschränkt wird die Empfehlung nur durch die noch nicht ganz ausgereifte Entwicklungsumgebung.

7. Schlussbetrachtung und Ausblick

7.1. Schlussbetrachtung

Das folgende Kapitel widmet sich der Schlussbetrachtung der Arbeit. Dabei wird festgehalten, was im Einzelnen erarbeitet wurde und welche Ergebnisse festzuhalten sind. Die Grundlage für die Arbeit bilden die theoretischen Grundlagen aus Kapitel 1. Veränderungen in dynamischen Märkten erfordern flexibel anpassbare IT-Systeme, was durch die Verwendung von Service-Orientierten Architekturen und Workflow Technologie vereinfacht wird.

Abläufe werden in Form von Geschäftsprozessen beschrieben, die eine Aneinanderreihung von Aktivitäten darstellen. In der im Fokus dieser Arbeit entwickelten Anwendung werden in Abschnitt 3.4 die Geschäftsprozesse für die Erstellung bzw. Verwaltung von Angeboten und Bestellungen beschrieben. Geschäftsprozesse werden im Laufe des Entwicklungsprozesses zu in BPEL beschriebenen Workflows, die Aktivitäten zu Web Services. Durch das Konzept der Service Component Architecture werden Workflows und Web Services abstrahiert, so dass ganze Anwendungen wie ein Service verwendet werden können. Dieses Konzept wird Software as a Service genannt und ermöglicht die Nutung derselben Softwareinstanz durch mehrere Mandanten.

Da die verschiedenen Mandanten unterschiedliche Anforderungen an eine Anwendung stellen, müssen bestimmte Teile selbiger variabel gehalten werden. Diese können mithilfe der Variabilitätsbeschreibungen aus Abschnitt 2.3.4 modelliert werden. Die explizite Modellierung vermeidet Inkonsistenzen, die auftreten können, wenn Variabilitäten über mehrere Anwendungsebenen hinweg mehrfach beschrieben werden. Patterns aus Abschnitt 2.3.3 verbessern die Verständlichkeit der Modellierung von mandantenfähigen Web Services. Die Variabilitäten der erstellten Anwendung werden in Abschnitt 3.5.2 modelliert und stellen Beispiele für die vorab beschrieben Patterns dar. Da die Variabilitäten einen fachlichen Bezug haben, werden sie innerhalb der Geschäftsprozesse beschrieben. Dies ermöglicht zusätzlich den Austausch und die Versionierung der Variabilitäten.

Zur Spezifikation einer Anwendung gehören auch Beschreibungen der Anwender und Anwendungsfälle, die in Abschnitt 3.2. beschrieben sind. Das Datenmodell aus Abschnitt 3.3 beschreibt alle verwendeten Geschäftsobjekte und deren Eigenschaften. Es dient als Grundlage für die Erstellung der Datenbanktabellen und die Generierung der Service Data Objects. Letztere werden von den Web Services verwendet, deren Implementierung in Abschnitt 4.3.1 beschrieben ist. Da auch die meisten anderen Komponenten der Anwendung automatisch generiert werden, reduziert sich der Implementierungsaufwand durch den verwendeten Entwicklungsprozess erheblich.

Ausschließlich die Web Services und die Benutzeroberfläche werden selbst implementiert. Für Letztere existieren mehrere Ansätze für eine automatische Generierung, die in Abschnitt 4.3.3 beschrieben sind. Leider erfüllt keiner dieser Ansätze typische Anforderungen an eine benutzerfreundliche Oberfläche, die zu der im Fokus dieser Arbeit erstellten Anwendung passt. Auch in anderen Bereichen, beispielsweise beim Test, unterstützt die verwendete Entwicklungsumgebung den Entwickler sehr gut. Alle nötigen Funktionen vom isolierten Testen bis hin zur Simulation ganzer Workflows werden angeboten.

Die erstellte Anwendung zeigt die Umsetzbarkeit von Variabilitätsbeschreibungen und den in Abschnitt 2.3.3 beschriebenen Patterns. In Abschnitt 5.3 wird dargelegt, warum die Wiederverwendung ganzer Workflows selten sinnvoll ist. Verschiedene Vorgehensweisen sind denkbar, um Veränderungen der Anwendung zu installieren, ohne den Betrieb zu unterbrechen. Diese Deployment Strategien sind in Abschnitt 6.1 beschrieben und mithilfe von Beispielen konkretisiert. Wie in den Abschnitten 6.2 und 6.3 beschrieben, haben die evaluierten Konzepte einen positiven Einfluss auf die Skalierbarkeit und Wartbarkeit von Anwendungen.

Da die im Fokus dieser Arbeit erstellte Anwendung einen Prototyp darstellt, sind die Erkenntnisse nur begrenzt auf produktiv eingesetzte Systeme anwendbar. Mögliche Problemquellen wie die Anbindung von Fremdsystemen werden aus Zeitgründen nicht genauer untersucht. Die in Abschnitt 6.2.1 beschriebenen Mängel der Entwicklungsumgebung haben Einfluss auf die Entwicklungskosten produktiv eingesetzter Systeme. Auch der Funktionsumfang der erstellten Anwendung ist stark eingeschränkt. Der folgende Abschnitt beschreibt mögliche Erweiterungen der Anwendung.

7.2. Ausblick

Verschiedene Erweiterungen der erstellten Anwendung sind denkbar. Naheliegend sind Funktionen zur Verwaltung von Fahrzeugen und Dienstleistungen. So kann ein Händler beispielsweise von ihm angebotene Dienstleistungen und Preise selbst pflegen. Die Konfiguration von Neufahrzeugen ist unentbehrlich für den produktiven Einsatz einer solchen Anwendung. Auch die Vereinfachungen im Datenmodell, der Berechnung der Preise und Geschäftsmodelle können abgebaut werden. Dazu gehören auch Händlerrabatte und die Unterstützung verschiedener Währungen. Wünschenswert ist eine Internationalisierung der Oberfläche, so dass abhängig vom angemeldeten Benutzer die entsprechende Landessprache verwendet wird.

Technische Erweiterungen wie das dynamische Binden von Web Services durch einen Enterprise Service Bus ermöglichen einen höheren Grad an Flexibilität der Software. Auch die Verwendung einer neueren Version der Laufzeitumgebung erhöht deren Funktionsumfang. Der inzwischen verfügbare WebSphere Process Server 6.2 [IBM4] bietet Funktionen, die mit bisherigen Versionen nicht möglich waren. Eine willkommene Verbesserung ist erweiterte Fallbearbeitung. Mit dieser ist es möglich, Human Tasks zu überspringen oder erneut auszuführen. Bei dem im Fokus dieser Arbeit erstellten Prototyp könnte damit die Zurück-Taste im Browser verwendet werden, um einen vorherigen Prozessschritt zu wiederholen. Abbildung 39 stellt diese Funktionalität schematisch dar.

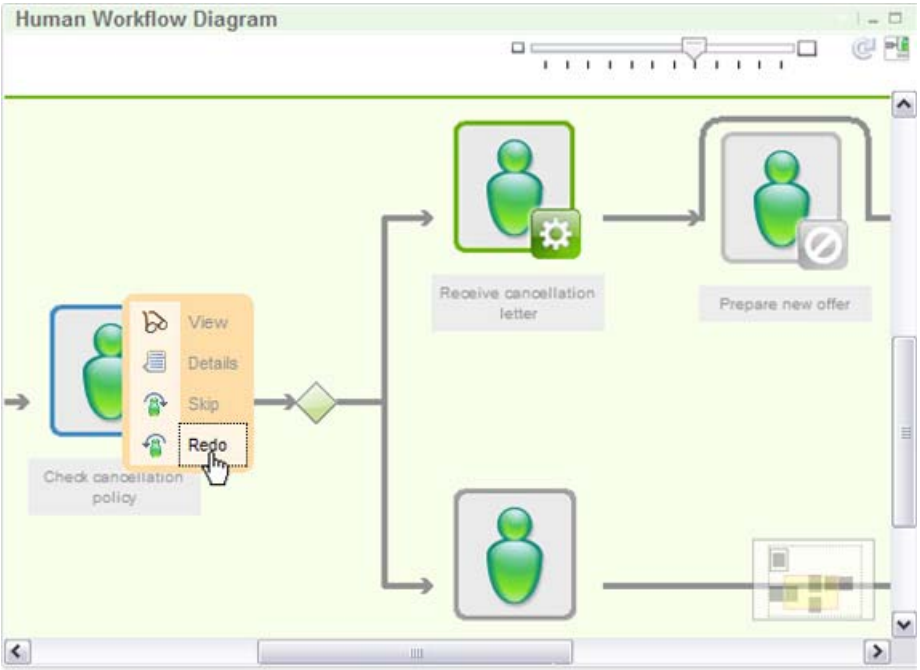

Abbildung 39: Neue Funktionalitäten im WPS 6.2 [IBM4]

Ein Beispiel für so genannte Widgets ist ebenfalls in Abbildung 39 dargestellt. Diese sind kleine Erweiterungen, die innerhalb einer Web 2.0 Oberfläche genutzt werden können. Viele Verbesserungen wurden im Bereich der Benutzeroberflächen eingearbeitet. Mit so genannten Business Spaces kann der Benutzer sich seine Oberfläche selbst zusammenstellen und seinen Wünschen entsprechend einsetzen.

An den in Abschnitt 4.2.2 beschriebenen Problemen bei der Synchronisierung zwischen Modellierungs- und Entwicklungsumgebung wurde gearbeitet. Dadurch ist es mit dem WebSphere Business Modeler 6.2 möglich, Workflows direkt aus der Modellierungsumgebung auf der Workflow Engine zu installieren. Die Entwicklungsumgebungen entwickeln sich also ähnlich dynamisch wie die Märkte, für die Anwendungen entwickelt werden. Man darf gespannt sein, wie diese Entwicklung weitergeht.

Literaturverzeichnis

AND08 Anderson, Chris: *The Long Tail - Why the Future of Business Is Selling Less of More*. New York, NY: Harper Collins 2008

CC06 Chong, Frederick; Carraro, Gianpaolo: *Architecture Strategies for Catching the Long Tail*. http://msdn.microsoft.com/en-us/library/aa479069.aspx

CHA04 Chappell, Dave: *Enterprise Service Bus*. Sebastopol, CA: O'Reilly 2004

CHA1 ChangeVision: *JUDE/Professional 5.4*. http://jude.change-vision.com/jude-web/product/professional.html

COC00 Cockburn, Alistair: *Writing Effective Use Cases*. Amsterdam: Addison-Wesley Longman 2000

FDF02 Fowler, Martin; Rice, David; Foemmel, Matthew u.a.: *Patterns of Enterprise Application Architecture*. Boston, MA: Pearson Education 2002

FIS01 Fischer, Layna (Bearb.): *Workflow handbook 2001*. Lighthouse Point, FL: Future Strategies 2001

GB06 Gaur, Harish; Blanvalet, Stany (Bearb.): *BPEL cookbook - best practices for SOA-based integration and composite applications development; ten practical real-world case studies combining business process management and web services orchestration*. Birmingham [u.a.]: Packt Publ. 2006

GFI1 GFI-iTmedia Consult GmbH: *DMS-Autoline-Projekte*. http://www.itmedia.de/geschaeftsfelder/dms-autoline-projekte/

HAV05 Havey, Michael: *Essential business process modeling*. Köln [u.a.]: O'Reilly 2005

HEI1 iX CeBIT 2009, S. 16: *Cloud Computing*. http://www.heise.de/ix/artikel/2009/cebit/016/

HFS06 Hantschel, Ralf; Ruf, Fabian; Strotbek, Haiko: *Vergleich von BPEL Laufzeitumgebungen*. Stuttgart, Univ., Studiengang Softwaretechnik, Fachstudie Nr. 54, 2006

HW03 Hohpe, Gregor; Woolf, Bobby: *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston, MA: Pearson Education 2003

IBM1 IBM: *DaimlerChrysler builds a "proactive infrastructure" to become more flexible and responsive*.
http://www-935.ibm.com/services/us/cio/pdf/bpt_cs_daimlerchrysler.pdf

IBM2 IBM: *BPELJ: BPEL for Java technology*.
http://www.ibm.com/developerworks/library/specification/ws-bpelj/

IBM3 IBM: *WS-BPEL Extension for People*.
http://www.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/

IBM4 IBM: *What's new in WebSphere Process Server V6.2*.
http://www.ibm.com/developerworks/websphere/bpmjournal/0812_fasbinder5/0812_fasbinder5.html

IBM5 IBM: *WebSphere Process Server*.
http://www-01.ibm.com/software/integration/wps/

IBM6 IBM: *WebSphere Integration Developer*.
http://www-01.ibm.com/software/integration/wid/

IBM7 IBM: *PAC-Studie: IBM behauptet SOA-Spitzenrang in Deutschland.*
http://www-05.ibm.com/de/pressroom/presseinfos/2009/02/20_2.html

IBM8 IBM: *Business Process Management auf der Grundlage von Smart SOA.*
http://www-01.ibm.com/software/de/solutions/soa/bpmsoa.html

IBM9 IBM: *DB2 Express Edition.*
http://www-01.ibm.com/software/data/db2/express/

IBM10 IBM: *Building SOA solutions with the Service Component Architecture.*
http://www.ibm.com/developerworks/websphere/techjournal/0510_brent/0510_brent.html

IBM11 IBM: *Configuring and using adapters.*
http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/index.jsp?topic=/com.ibm.wbit.612.help.adapter.emd.ui.doc/topics/tcreatecmps.html

IBM12 IBM: *Customizing WebSphere Application Server Enterprise Process Choreographer Web Client.*
http://www.ibm.com/developerworks/websphere/library/techarticles/wasid/WPC_Client1/WPC_Client1.html

IBM13 IBM: *Business Process Management Samples & Tutorials.*
http://publib.boulder.ibm.com/bpcsamp/v6r1/index.html

IBM14 IBM: *Dokumentation für IBM WebSphere Integration Developer.* http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/index.jsp?topic=/com.ibm.wbit.612.help.nav.doc/topics/welcome.html

IBM15 IBM: *Building a human task-centric business process with WebSphere Process Server.* http://www.ibm.com/developerworks/websphere/library/techarticles/0702_shankar/0702_shankar.html

IJC07 Ashok Iyengar, Vinod Jessani, and Michele Chilanti: *WebSphere business integration primer: Process server, BPEL, SCA, and SOA*. Boston, MA: Pearson Education 2007

LR00 Leymann, Frank; Roller, Dieter: *Production workflow: concepts and techniques*. Upper Saddle River, NJ: Prentice Hall PTR 2000

ML08a Mietzner, Ralph; Leymann, Frank: *Generation of BPEL Customization Processes for SaaS Applications from Variability Descriptors.* In: Proceedings of the IEEE 2008 International Conference on Services Computing (SCC 2008), Industry Track, Honolulu, HI, USA 2008

ML08b Mietzner, Ralph; Leymann, Frank: *Towards Provisioning the Cloud: On the Usage of Multi-Granularity Flows and Services to Realize a Unified Provisioning Infrastructure for SaaS Applications.* In: Proceedings of the IEEE 2008 Congress on Services (SERVICES 2008), Honolulu, HI, USA 2008

MLP08 Mietzner, Ralph; Leymann, Frank; Papazoglou, Mike: *Defining Composite Configurable SaaS Application Packages Using SCA, Variability Descriptors and Multi-Tenancy Patterns.* In: Proceedings of the 3rd International Conference on Internet and Web Applications and Services (ICIW 2008), Athens, Greece 2008

MML09 Mietzner, Ralph; Metzger, Andreas; Leymann, Frank u.a.: *Variability Modeling to Support Customization and Deployment of Multi-Tenant-Aware Software as a Service Applications.* In: Proceedings of ICSE 2009 Workshop - Principles of Engineering Service Oriented Systems (PESOS), Vancouver, Kanada 2009

MUT09 Mietzner, Ralph; Unger, Tobias; Titze, Robert u.a.: *Combining Different Multi-Tenancy Patterns in Service-Oriented Applications.* Under submission

OAS1 OASIS: *UDDI Specification.* http://uddi.xml.org/specification

OAS2 OASIS: *Web Services Business Process Execution Language*. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

OMG1 OMG: *Business Process Modeling Notation*. http://www.omg.org/spec/BPMN/1.2/

OMG2 OMG: *Unified Modeling Language (UML)*. http://www.omg.org/spec/UML/

OSL06 Overdick, Hagen; Schnieders, Arnd; Lindenthal, Wieland: *Process Family Model Driven Software Generator: A Case Study in the E-Business Domain*. http://www.pesoa.de/pages/Publications/DemosPresentations/Magrathea/Magrathea_Presentation.ppt

OSO1 Open SOA: *Service Component Architecture Specifications*. http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications

OSO2 Open SOA: *Service Data Objects Specifications*. http://www.osoa.org/display/Main/Service+Data+Objects+Specifications

PAR72 Parnas, David L.: *On the criteria to be used in decomposing systems into modules*. In: Communications of the ACM, New York, NY, USA 1972

PM08 Pohl, Klaus; Metzger, Andreas: *Variabilitatsmanagement in Software-Produktlinien*. In: Proceedings of Software Engineering (SE 2008), Munich, Germany 2008

SC02 Seltsikas, Philip; Currie, Wendy: *Evaluating the Application Service Provider (ASP) Business Model: The Challenge of Integration*. In: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02), Volume 8, Honolulu, HI, USA 2002

SCH08 Schumm, David: *Graphische Modellierung von BPEL Prozessen unter Verwendung der BPMN Notation*. Stuttgart, Univ., Studiengang Informatik, Diplomarbeit Nr. 2720, 2008

SUN1 Sun Microsystems: *Java SE Technologies - Database*. http://java.sun.com/javase/technologies/database/index.jsp

TEO05 Teorey, Toby J.: *Database Modeling and Design - Logical Design*. Heidelberg [u.a.]: Morgan Kaufmann 2005

TIG1 tigris.org: *TortoiseSVN*. http://tortoisesvn.tigris.org/

W3C1 W3C: *Web Services Activity*.
http://www.w3.org/2002/ws/

W3C2 W3C: *Web Services Glossary*.
http://www.w3.org/TR/ws-gloss/

W3C3 W3C: *Extensible Markup Language (XML) 1.0 (Fifth Edition)*.
http://www.w3.org/TR/REC-xml/

W3C4 W3C: *SOAP Version 1.2*.
http://www.w3.org/TR/soap/

W3C5 W3C: *HTTP - Hypertext Transfer Protocol*.
http://www.w3.org/Protocols/

W3C6 W3C: *Web Services Description Language (WSDL) 1.1*.
http://www.w3.org/TR/wsdl

WCL05 Weerawarana, Sanjiva; Curbera, Francisco; Leymann, Frank u.a.: *Web services platform architecture - SOAP, WSDL, WS-policy, WS-addressing, WS-BPEL, WS-reliable messaging and more*. Upper Saddle River, NJ [u.a.]: Prentice Hall PTR 2005

Alle aufgeführten Links wurden zuletzt am 22.04.2009 überprüft.

Abbildungsverzeichnis

Abbildung 1: SOA-Dreieck [WCL05] 9
Abbildung 2: SOA-Lifecycle [IJC07] 10
Abbildung 3: Workflow Dimensionen [LR00] 12
Abbildung 4: Workflow-zentrierte Applikation [HAV05] 13
Abbildung 5: SCA-Komponente [IJC07] 15
Abbildung 6: SaaS-Architektur [CC06] 17
Abbildung 7: Service Patterns [MUT09] 19
Abbildung 8: Invocation Patterns [MUT09] 20
Abbildung 9: Orthogonale Variabilitätsmodellierung [PM08] 22
Abbildung 10: Geschäfts- und Bewilligungszustände 27
Abbildung 11: Anwendungsfälle 29
Abbildung 12: Datenmodell 33
Abbildung 13: Legende Geschäftsprozesse 36
Abbildung 14: Angebotsprozess 37
Abbildung 15: Geschäftsprozess Kunde identifizieren 37
Abbildung 16: Geschäftsprozess Fahrzeug auswählen 38
Abbildung 17: Geschäftsprozess Angebot erstellen 38
Abbildung 18: Bestellprozess 39
Abbildung 19: Geschäftsprozess Bestellung verschicken 39
Abbildung 20: Geschäftsprozess Bestellung prüfen 40
Abbildung 21: Geschäftsprozess Angebote verwalten 41
Abbildung 22: Geschäftsprozess Bestellungen verwalten 42
Abbildung 23: Variabilität Dienstleistungen auswählen 43
Abbildung 24: Variabilität Gesamtpreisberechnung 44
Abbildung 25: Variabilität Angebot bestätigen 44
Abbildung 26: Variabilität Kreditwürdigkeit prüfen 45
Abbildung 27: Architekturüberblick 47
Abbildung 28: Workflow im WID 49
Abbildung 29: SCA-Diagramm 50
Abbildung 30: Synchronisierungsfenster 51
Abbildung 31: generierte Oberfläche 57
Abbildung 32: Single Person Workflow Konzept [IBM13] 58
Abbildung 33: Single Person Workflow Oberfläche 59
Abbildung 34: Modulübersicht 60
Abbildung 35: BPCE - Verwaltung von Prozessinstanzen 62
Abbildung 36: Web Service Test 63
Abbildung 37: Regressionstest 64
Abbildung 38: BPCE - Verwaltung von Human Tasks 65
Abbildung 39: Neue Funktionalitäten im WPS 6.2 [IBM4] 75

Tabellenverzeichnis

Tabelle 1: BPEL Hauptbestandteile 14
Tabelle 2: Dienstleistungsklassen..... 28
Tabelle 3: Anwendungsfall Benutzer anmelden..... 30
Tabelle 4: Anwendungsfall Kunde identifizieren..... 30
Tabelle 5: Anwendungsfall Fahrzeug auswählen..... 31
Tabelle 6: Anwendungsfall Angebot erstellen 31
Tabelle 7: Anwendungsfall Bestellung verschicken 31
Tabelle 8: Anwendungsfall Bestellung prüfen 32
Tabelle 9: Anwendungsfall Angebote verwalten 32
Tabelle 10: Anwendungsfall Bestellungen verwalten..... 32
Tabelle 11: Systemtest Beispiele..... 67

Abkürzungsverzeichnis

ASP	Application Service Provider
BPEL	Business Process Execution Language
BPCE	Business Process Choreographer Explorer
BPM	Business Process Management
BPMN	Business Process Modeling Notation
CRM	Customer Relationship Management
EAI	Enterprise Application Integration
ER	Entity Relationship
eCCo	eCommerce Concept
ESB	Enterprise Service Bus
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IAAS	Institut für Architektur von Anwendungssystemen
JDBC	Java Database Connectivity
JSP	Java Server Page
SaaS	Software as a Service
SCA	Service Component Architecture
SDO	Service Data Objects
SLA	Service Level Agreement
SOA	Service-Orientierte Architektur
UML	Unified Modeling Language
UTC	Universal Test Client
W3C	World Wide Web Consortium
WBM	WebSphere Business Modeller Advanced
WID	WebSphere Integration Developer
WPS	WebSphere Process Server
WSDL	Web Service Description Language
XML	Extensible Markup Language
XSD	XML Schema Definition

Inhalte der CD-ROM

Die zur Arbeit gehörende CD-ROM enthält folgende Inhalte:

- Diplomarbeit
  - Abbildungen
  - Dokumentation
  - Literatur
  - Prozesse
  - Diplomarbeit als PDF Datei
- WebSphere Business Modeller Workspace
- WebSphere Integration Developer Workspace

Erklärung

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

Ort, Datum Unterschrift