Integration von Nutzungsdaten heterogener Dienstplattformen: Eine Webservice-orientierte Architektur

Diplomarbeit

Ayhan Kondoz
Fachbereich STS

Technische Universität Hamburg Harburg

28. Juli 2005

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

1.1. Motivation

Zu den betrieblichen Grundentscheidungen innerhalb eines Unternehmens gehört an einer der vorderen Stellen die Festsetzung des Unternehmensziels. Das Unternehmensziel gibt vor, nach welchen Entscheidungsregeln die Unternehmung handeln soll. Diese Ziele werden in der Regel von den Gründern und Kapitalgebern festgesetzt. Die Zielsetzung ist untergliedert in Oberziele, Zwischenziele und Unterziele.

Gewinnmaximierung ist oftmals eines der Oberziele von Unternehmen. Andere Oberziele sind unter anderem die Unternehmenserhaltung, Wirtschaftlichkeit und langfristige Einzelkundenbindung.

Das eigentliche Ziel des Unternehmens ist allerdings die langfristige Gewinnmaximierung bzw. Gewinnoptimierung, also über einen langen Zeitraum möglichst hohe Gewinne mit Blick auf die langfristige Unternehmenserhaltung zu erwirtschaften. Wenn ein Unternehmen möglichst kurzfristig einen maximalen Gewinn erzielen wollte, bräuchte es nur den Verkauf anzustreben.

Es gibt verschiedene Gliederungsmöglichkeiten des Unternehmenszieles.

- Vertikale Gliederung: Die vertikale Gliederung in Ober-, Zwischen- und Unterziele (Teilziele) dient der Rangordnung der Ziele. Oberziele sollen durch die jeweils untergeordneten Ziele erreicht werden.
- Horizontale Gliederung: Einteilung der Ziele in verschiedene Kategorien

  - leistungswirtschaftlich, z.B. Marktanteile
  - wachstumsbezogen, z.B. Erschliessung neuer Märkte
  - sozial, z.B. Schaffung von Arbeitsplätzen
  - ökologisch, z.B. Umweltschutz
  - güterwirtschaftlich, z.B. hohes Qualitätsniveau
  - führungsbezogen, z.B. gute Mitarbeiterführung
  - finanzwirtschaftlich, z.B. Rentabilität, Liquidität, Sicherheit, Unabhängigkeit von Geldgebern
- Darüber hinaus gibt es auch quantitative, d.h. in Zahlen messbare und qualitative Unternehmensziele. Beispiele:
  - qualitativ: minimaler Ressourcenverbrauch
  - quantitativ: Erreichen eines Marktanteils von x Prozent

1.2. Ziel der Arbeit

Die freenet.de AG ist ein Dienstleistungsunternehmen, das seinen Kunden eine Vielzahl von verschiedenen Dienstleistungen anbietet. Einige der Folgen, die durch die Verfolgung der Unternehmensziele entstehen sind:

- steigende Anzahl von Kunden
- wachsende Komplexität der angebotenen Dienstleistungen
- steigende Anzahl von angebotenen Dienstleistungen
- ...

Als Reaktion auf dieses Verhalten werden Komponenten, die von mehreren Anwendungen verwendet werden, als zentrale Dienstleistungen innerhalb der freenet.de AG Systemlandschaft eingerichtet. Die Stammdatenverwaltung der Kunden ist sicherlich das beste Beispiel. Wenn jede Anwendung eine eigene Stammdatenverwaltung hat, müssen die Kundendaten durch alle Anwendungen hindurch synchronisiert werden, was Ressourcen verschwendent, fehleranfällig und uneffizient ist. Durch eine zentrale Verwaltung der Stammdaten treten diese Problemfälle erst garnicht auf.

Das Ziel dieser Diplomarbeit ist ein weiterer Schritt in der Zentrallisierung von systemübergreifenden Komponenten. Einige der von der freenet.de AG angebotenen Dienstleistungen wie DSL Flatrate und E-Mail Premium[1] benötigen kein Accounting von Verbrauchsdaten, wohingegen andere Dienstleistungen wie SMS, Online-Content[2] und Musikdownload eine Nutzungsdatenverwaltung benötigen. Der Kunde erwirbt bei diesen Dienstleistungen ein bestimmtes (pre-paid[3]) Kontingent und kann den angebotenen Service so lange in Anspruch nehmen bis dieses Kontingent ausgeschöpft ist. D.h. die Nutzungsdaten und Kontingente dieser Kunden müssen verwaltet werden.

Ein weiterer guter Grund für einen zentralen Dienst zur Nutzungsdatenverwaltung ist, dass es ermöglicht werden soll, dass verschiedene Anwendungen mit den gleichen Konten arbeiten. Der SMS Versand[4] zum Beispiel sollte aus verschiedenen Applikationen heraus verwendtbar sein. Dieses ist bei einer Nutzungsdatenverwaltung auf Applikationsebene nicht möglich bzw. birgt viele Fehlerquellen, weil die Applikationen untereinander synchronisiert werden müssten.

Durch die Einführung eines zentralen Services zur Integration von Nutzungsdaten heterogener Dienstplattformen werden:

1. Ressourcen gespart

[1] E-Mail Premium: Verschiedene Dienstleistungen rund um das E-Mail Office wie SMS, Domainname, Fax usw.
[2] Online-Content: Artikel, Texte oder sonstiger content, der bezahlpflichtig ist.
[3] pre-paid: Die Bezahlung findet vor der Auslieferung der Dienstleistung statt.
[4] SMS Versand: Ein Online-Dienst der freenet.de AG zum Verschicken von SMS-Nachrichten.

2. Entwicklungszeiten reduziert
3. Fehlerquellen minimiert
4. Kundenfreundlichkeit maximiert
5. Administrationsaufwand minimiert

1.3. Aufbau der Arbeit

In **Kapitel 1** wird die Motivation und die Ziele der Arbeit dargestellt und ein Überblick über die Arbeit gegeben.

In **Kapitel 2** wird die Systemumgebung der freenet.de AG vorgestellt und ein Prozessablauf und die Systemlandschaft dokumentiert.

In **Kapitel 3** wird das Ziel dieser Arbeit und der Lösungsweg, um dieses Ziel zu erreichen, erarbeitet.

In **Kapitel 4** werden die Vorgaben und Einschränkungen, die von der freenet.de AG gestellt wurden, beschrieben.

In **Kapitel 5** wird die eingesetzte Architektur bestimmt und ein Entwurf des Systems erstellt.

In **Kapitel 6** werden die einzelnen Schritte der Implementierung untersucht und beschrieben.

In **Kapitel 7** wird die Arbeit zusammengefasst und es wird ein Ausblick auf Probleme und ToDos die aufkommen könnten gegeben.

Im **Anhang A** wird eine Einführung in die wichtigsten Komponenten der freenet.de AG Systemlandschaft gegeben.

Im **Anhang B** werden Tools und Techniken, die bei der Implementierung dieser Arbeit zum Einsatz gekommen sind, dokumentiert.

2. Systemumgebung der freenet.de AG

Als Einführung in das Thema der Nutzungsdatenverwaltung ist es zuerst wichtig, die Systemumgebung kennen zu lernen, in der die Nutzungsdatenverwaltung zum Einsatz kommen wird. Das folgende Kapitel wird zu diesem Zweck eine Einführung in die Systemlandschaft im Rahmen eines Standart-Bestellprozesses geben und den Prozessablauf einer Bezahlbeziehung zwischen einem Kunden und der freenet.de AG untersuchen.

2.1. Prozessablauf einer Bezahlbeziehung

Wie genau sieht ein Bestellprozess aus? Was für Komponenten nehmen an diesem Prozessablauf teil und wie interagieren sie miteinander? Diese Fragen werden im folgenden beantwortet.

Für ein leichteres Verständnis wird der Prozessablauf in zwei Blöcke unterteilt und betracht. Der erste Block besteht hierbei in der Aufnahme und Verarbeitung einer Bestellung bis der Vertrag im Billingsystem [1] angelegt ist. Der zweite Block hingegen beinhaltet die Prozesse, die vom Billingsystem angestossen werden wie z.B. der Rechnungsstellung.

[1] Lean NextGen [8]: Billingsystem zur Abrechnung von Verbrauchsdaten und Erstellung von Rechnungen.

2.1.1. Bestellungsaufnahme

Der folgende Prozessablauf stellt eine generalisierte Bestellungsaufnahme mit den daran teilnehmenden Kernelementen dar. Dieser Idealfall liegt in der Praxis allerdings nur selten vor. In der Praxis sind die Prozessabläufe für verschiedene Produkte sehr oft (leicht) unterschiedlich und es müssen in Abhängigkeit der Produkte Sonderbehandlungen eingeführt werden.

Man vergleiche zum Beispiel die zwei Bestellprozesse für DSL und iPhone[2]. Technisch gesehen sind die beiden Produkte sehr ähnlich. Der Kunde kauft sich die Bereitstellung eines netzseitigen Services. Vom Standpunkt der Kunden gibt es allerdings einen gravierenden Unterschied. Ein DSL bestellender Kunde erwartet keine augenblickliche Freischaltung. Obwohl dieses technisch möglich ist, ist der Kunde auch mit einer Wartezeit durchaus zufrieden. Deshalb werden alle weiteren Schritte, nachdem die Bestellung eingegangen ist, asynchron verarbeitet. Bei einer iPhone Bestellung hingegen erwartet ein Kunde, dass er sofort, nachdem er die Bestellung aufgegeben hat, dieses Produkt auch benutzen kann. Aus diesem Grund muss das Anlegen und Freischalten des Kunden im iPhone-Bestellprozess synchron bei der Eingabe der Bestellung getätigt werden.

Die Betrachtung solcher produktabhängigen Sonderfälle tragen mit Blick auf das Ziel dieser Diplomarbeit allerdings keine Relevanz und werden deshalb vernachlässigt.

Der Basic Flowchart des Prozessablaufes der Bestellungsaufnahme ist in Abbildung 2.1 dargestellt. Die Reihenfolge in dem die einzelnen Prozessschritte durchlaufen werden kann anhand der Nummerierung nachverfolgt werden.

1. Produktauswahl:
   Am Anfang muss der Kunde auswählen welche Produkt(e) er bestellen möchte. In den meisten Fällen geschieht dies, indem sich der Kunde in einem Internet-Shop die Produkte mit Hilfe eines Internet Browser zusammensucht.

[2]iPhone: Telefonie-Dienst über voice-over-IP Technologie.

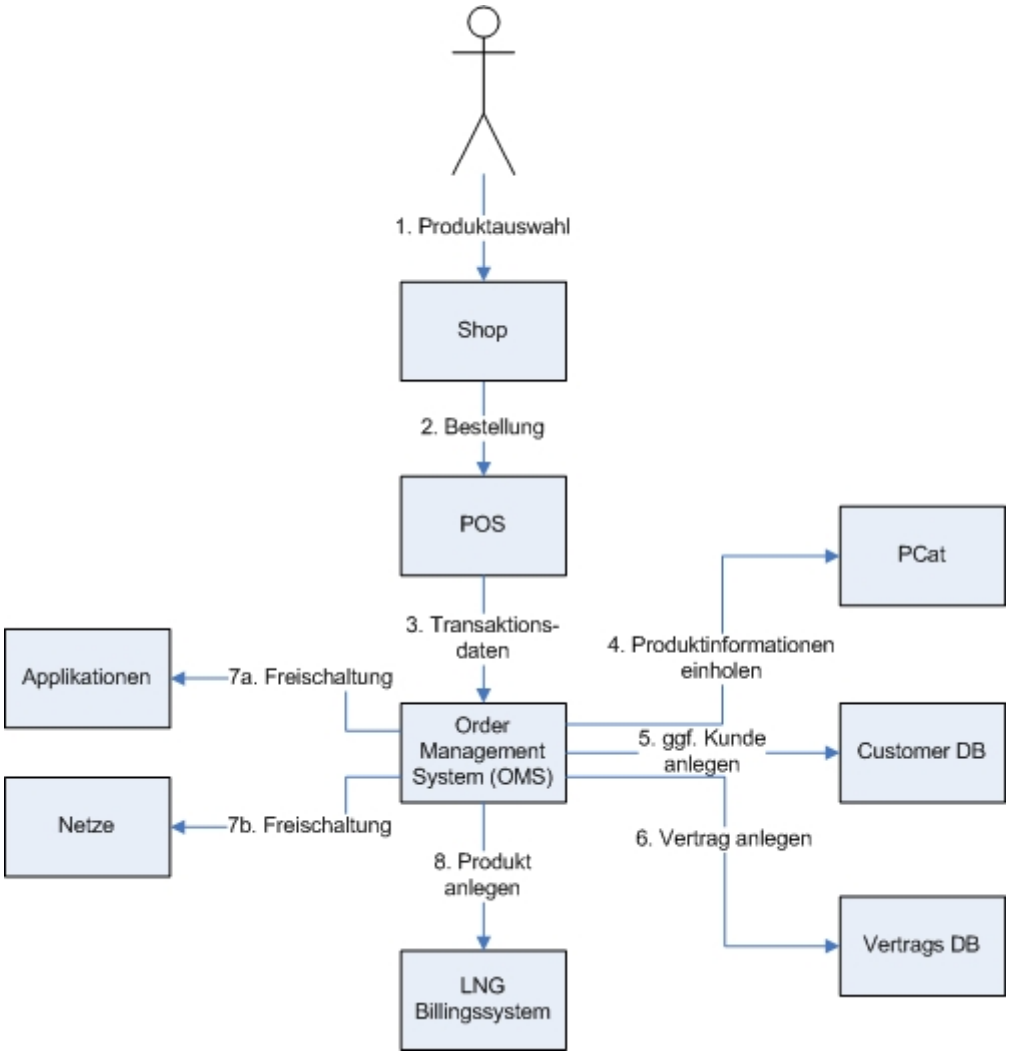

Abbildung 2.1.: Bestellungsaufnahme eines Standart Bestellprozesses

2. Bestellung:
   Nachdem die Auswahl der Produkte abgeschlossen ist, wird der Kunde auf den so genannten Point Of Sale (POS)[3] weitergeleitete. Der Point Of Sale ist dafür zuständig, jegliche für die erfolgreiche Ausführung der Bestellung benötigten Kundeninformationen aufzunehmen. Welche Informationen dieses

[3]POS: Einzelheiten zum Point Of Sale sind im Anhang A.1 zu finden.

im Einzelnen sind, hängt von den bestellten Produkten ab. Die Stammdaten (Name, Adresse, PLZ, Ort etc.) sowie die Bezahlinformationen (Bank & Kontonummer oder Institut & Kreditkartennummer) sind Pflichtinformationen.

Nachdem der Kunde seine Bestellung erfolgreich eingegeben hat, ist der interaktive Teil des Bestellprozesses beendet. Die danach folgenden Prozesspunkte werden asynchron abgearbeitet.

3. Transaktionsdaten:
   Die Transaktionsdaten werden vom POS zusammengestellt und via eines SOAP Aufrufes an das Order-Management-System (OMS)[4] übergeben. Das Order-Management-System ist eine workflow System das von der freenet.de AG entwickelt wurde und Aufgaben wie die Prozesssteuerung (z.B. Auftragsabwicklung) und System-Integrationsaufgaben übernimmt.

   Die Transaktionsdaten beinhalten alle Daten, die vom Kunden eingegeben oder im Bestellprozess festgelegt wurden und für die erfolgreiche Durchführung der Bestellung benötigt werden.

4. Produktinformationen einholen:
   Das Order-Management-System muss die erhaltenen Transaktionsdaten mit Informationen vom Produkt-Katalog (PCat)[5] anreichern. Die Transaktionsdaten enthalten lediglich eine oder mehrere Produkt ID(s). Weitere benötigten Produktinformationen über Preis, Service Type[6], Service ID[7] usw. werden vom Produktkatalog eingeholt. Der Produktkatalog ist ein zentraler Dienst, der systemübergreifende Produktinformationen wie Preise, Texte, Abhängigkeiten, Service Types usw. verwaltet und wird im Anhang A.4 näher beschrieben.

[4] OMS: Eine Beschreibung des Order-Management-Systems finden Sie im Anhang A.3.
[5] PCat: Einzelheiten zum Produktkatalog befinden sich im Anhang A.4
[6] Service Type: Identifier für das Billingsystem zum Festlegen, um welchen Type von Service es sich handelt (z.B. Content, DSL, iPhone usw.).
[7] Service ID: Identifier für das Billingsystem zum Festlegen der Service ID. Die Service ID zusammen mit dem Service Type identifiziert ein Produkt eindeutig im Billingsystem.

5. Ggf. Kunde anlegen:
   Für den Fall, dass kein login Bereitgestellt wurde und es sich um einen Neukunden handelt, muss für den Kunden ein login auf der Kundendatenbank (customer DB) angelegt werden. Die Kundendatenbank enthält die Stammdaten sowie Bezahlinformationen für alle freenet.de AG Kunden. Der Name impliziert, dass es sich um eine Datenbank handelt, auf die Kunden oder Applikationen direkt Zugreifen können, dies ist aber nicht der Fall. Der Zugriff auf das System geschieht über einen SOAP-Server. Anhang A.2 gibt eine detailliertere Beschreibung des Customer Managers.

6. Vertrag anlegen:
   Nachdem nun der Kunde angelegt und die Transaktionsdaten angereichert wurden wird der Vertrag in der Vertrags-DB angelegt.

7. Freischaltung:
   In Abhängigkeit davon welches Produkt der Kunde gerade erworben hat ist es in den meisten Fällen nötig die zu dem Produkt gehörigen Dienstleistungen für den Kunden freizuschalten.

8. Produkt anlegen:
   Abschliessend wird der Kunde im Billingsystem [8] angelegt und dem Kunden das Produkt zugeordnet, damit der Verbrauch angemessen in Rechnung gestellt werden kann. Weitere Informationen zum Billingsystem befinden sich im Anhang A.5.

Nachdem das Produkt für den Kunde im Billingsystem angelegt wurde, ist die Bestellungsaufnahme abgeschlossen.

2.1.2. Rechnungsstellung

Der zweite Teil des Bestellprozesses, also die Rechnungsstellung, wird erreicht, nachdem der Kunde erfolgreich im Billingsystem angelegt wurde. Im Gegensatz zur Bestellungsaufnahme wird die Rechnungsstellung mehrmals, mindestens jedoch

einmal, durchlaufen. Sobald dieser Prozessabschnitt erreicht ist, sind die zeitkritischen Aufgaben ausgeführt. Zeitkritisch in dem Sinn, dass der Kunde ein Produkt bestellt hat und das erworbene Produkt oder die Dienstleistung so schnell wie möglich erhalten möchte. Die folgenden Schritte der Rechnungsstellung sind aus Sicht des Kunden sekundär bzw. eine Verzögerung wird durch die Kunden eher positiv als negativ aufgenommen.

Abbildung 2.2 zeigt den Basic Flowchart der Rechnungsstellung. Wie im vorherigen Kapitel ist die Reihenfolge, in dem die einzelnen Prozessschritte durchlaufen werden, anhand der Nummerierung verdeutlicht.

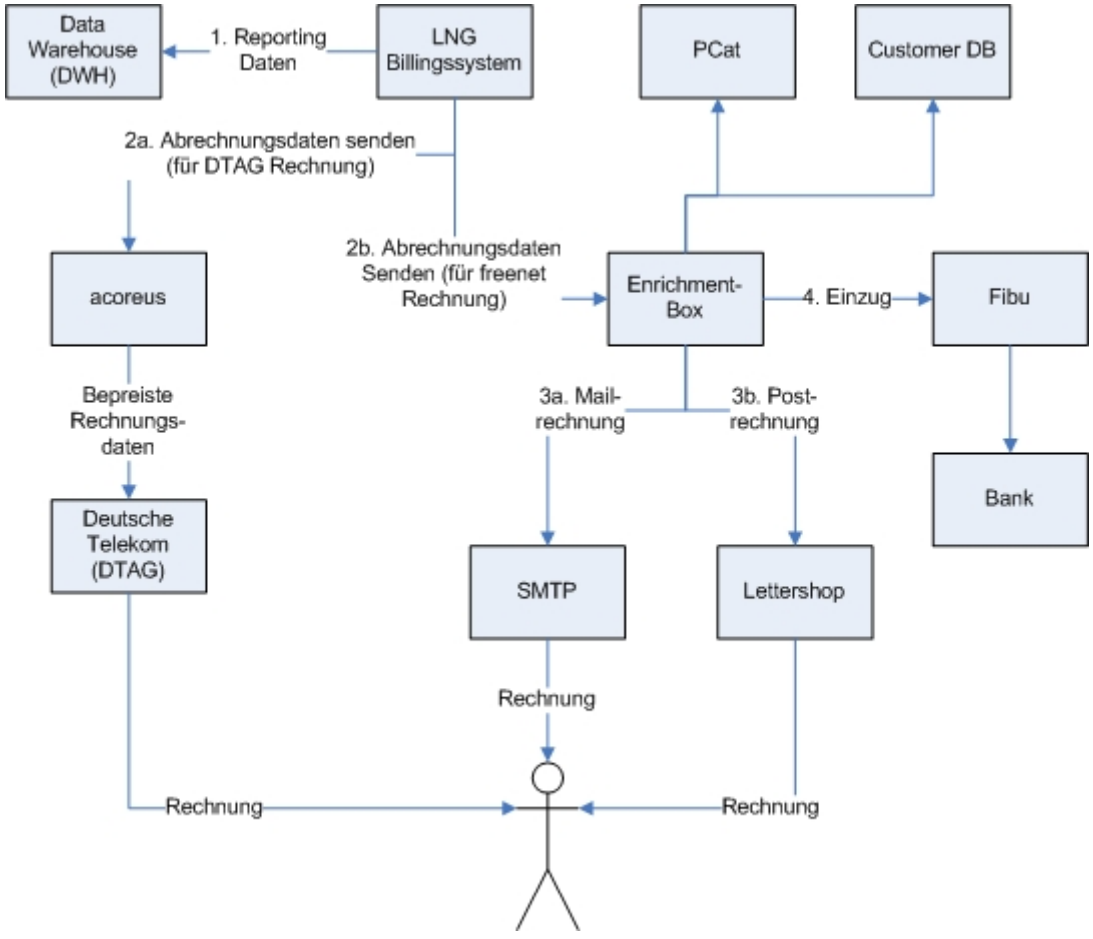

Abbildung 2.2.: Prozessablauf der Rechnungsstellung

1. Reporting-Daten:
   Die Verbrauchsdaten werden einmal täglich vom Billingsystem an das Data Warehouse übermittelt. Das Data Warehouse bereitet die Daten auf und stellt Tools zur Verfügung, um zum Beispiel Auswertungen für das Controlling und das Produkt Management zu generieren.

2. Abrechnungsdaten senden:
   Zurzeit werden zwei Arten von Rechnungen erzeugt. Zum einen werden die Beträge über die Deutsche Telekom (dtag) Rechnung abgebucht und zum anderen über die freenet.de AG eigene Rechnungsstellung.

   Die Rechnungsstellung über die Telekom wird mit Hilfe einer externen Firma (acoreus[8]) gestellt. Zu diesem Zweck werden die Abrechnungsdaten an acoreus geschickt, wo sie auf das von der Telekom vorausgesetzte Format gebracht und an die Telekom weitergereicht werden. Die Telekom ist dafür zuständig, die Beträge auf den Telekom-Rechnungen auszuweisen, die Beträge von den Kunden einzufordern und an die freenet.de AG weiter zu reichen.

   Die freenet.de AG eigene Rechnung wird mit Hilfe einer Eigenentwicklung, die Enrichment-Box genannt wird, erzeugt. Die Beträge werden mit Hilfe des Finanzbuchhaltungssystems von den zuständigen Instituten eingefordert.

3. Rechnung versenden:
   Die erste Aufgabe der Enrichment-Box ist es mit den vom Billingsystem übertragenen Daten eine Rechnung zu erstellen und dem Kunden zukommen zu lassen. Dazu müssen die Daten mit Hilfe der Kundendatenbank und des Produktkataloges angereichert werden. Von der Kundendatenbank werden der Name, die Adresse, die E-Mail Adresse usw. ermittelt und aus dem Produkt-Katalog der Produktname und eventuell vorhandene weitere Produkttexte, die auf der Rechnung erscheinen sollen.

   Je nachdem, ob der Kunde seine Rechnung per E-Mail und / oder per Post

[8] acoreus: Externer Partner, der mit der Deutschen Telekom (dtag) zusammenarbeitet.

erhalten möchte, wird die Rechnung per SMTP an den Kunden oder als XML Datei an den Lettershop[9] geschickt.

4. Einzug:
   Abschliessend wird die Finanzbuchhaltung über die Rechnungsstellung informiert. Die Aufgaben des Finanzbuchhaltungssystemes sind unter anderem, dass die Rechnungsbeträge von den zuständigen Instituten eingefordert und die Zahlungen überwacht werden, damit auftretende Probleme so schnell wie möglich verarbeitet werden können. Das Finanzbuchhaltungssystem unterstützt das Lastschriftverfahren sowie Kreditkarten.

2.2. Systemlandschaft einer Bezahlbeziehung

In der folgenden groben Skizzierung der Systemlandschaft sind einige der im Kapitel 2.1 für ein besseres Verständnis eingeführten Details vernachlässigt. Der Vollständigkeit halber wurden diese Details des Bestellprozesses zwar eingeführt, sie sind jedoch für das weitere Verständnis, das im Rahmen dieser Diplomarbeit benötigt wird, nicht weiter von Belangen. Zum Beispiel werden die Einzelheiten der Rechnungsstellung zu einem einzigen Prozessschritt zusammengefasst. Es ist ausreichend zu wissen, dass dieser Prozessschritt aus mehreren Einzelschritten zusammengesetzt ist.

Abbildung 2.3 zeigt die Systemlandschaft am Beispiel des E-Mail Office, das eine eigene Nutzungsdatenverwaltung für die SMS-Produkte einsetzt. Anstatt der generalisierten Systemlandschaft wird ein konkretes Produkt als Beispiel genutzt, damit es einfacher ist, die Zusammenhänge zu verstehen.

Die Parallelen zum Kapitel 2.1 sind bei genauer Betrachtung leicht ersichtlich. Der Kunde gibt am Point Of Sale[10] einen Auftrag ein, welcher an das Order Mana-

[9] Lettershop: Externe Druckerei, an den die Daten als XML Dateien übermittelt werden, der Serienbriefe erzeugt, druckt, in Umschläge steckt und verschickt.
[10] Point Of Sale: Siehe Anhang A.1

gement System[11] zur Verarbeitung weitergereicht wird. Das Order Management System führt alle nötigen Schritte aus, um den Auftrag auszuführen und das Billingsystem schreibt die Rechnungen.

Das Augenmerk liegt auf der Interaktion zwischen dem Order Management System und dem E-Mail Office, sowie zwischen dem E-Mail Office und dem Billingsystem. Bei einer Gegenüberstellung mit dem in Abbildung 2.1 dargestellten Prozess, wäre dies der Prozessschritt 7, also die Freischaltung, sowie die Konsequenz davon.

In dem hier benutzten Beispiel mit dem "SMS Packet" hat der Kunde das Recht erworben, monatlich eine bestimmte Anzahl von SMS-Nachrichten zu verschicken. Das Order Management System muss somit bei der Verarbeitung der Bestellung im E-Mail Office das Recht darauf, SMS-Nachrichten zu verschicken, freischalten. Das E-Mail Office übernimmt die Nutzungsdatenverwaltung der SMS-Nachrichten sowie die Benachrichtigung des Billingsystem über den Verbrauch, die vom Kunden verursacht wurden.

Der wichtige Satz des Vorherigen Absatzes lautet: *"Die Applikation übernimmt die Nutzungsdatenverwaltung"*. Auf den ersten Blick sieht das wie eine gute Strategie aus. Was aber passiert, wenn zwei oder mehr Applikationen die gleichen Nutzungsdaten verwenden möchten? D.h. am Beispiel der SMS-Nachrichten, wenn dem Kunden die Möglichkeit gegeben werden soll, innerhalb weiterer Applikationen und / oder Clients SMS-Nachrichten zu verschicken. Die verschiedenen Applikationen müssten sich untereinander synchronisieren und dafür sorgen, dass Kontingente, die dem Kunden zur Verfügung gestellt wurden, nicht überschritten werden. Ausserdem gibt es mehrere verschiedene Einheiten, wie zum Beispiel SMS, EUR, iPhone Pre-Paid Minuten und Upload- bzw. Downloadlimits, um nur einige zu nennen.

Fazit: Die Nutzungsdatenverwaltung auf Applikationsebene ist für eine einzelne separierte Applikation einsetzbar, birgt jedoch viele Gefahren und Probleme in einem System von heterogenen Dienstplattformen.

[11] Order Management System: Siehe Anhang A.3

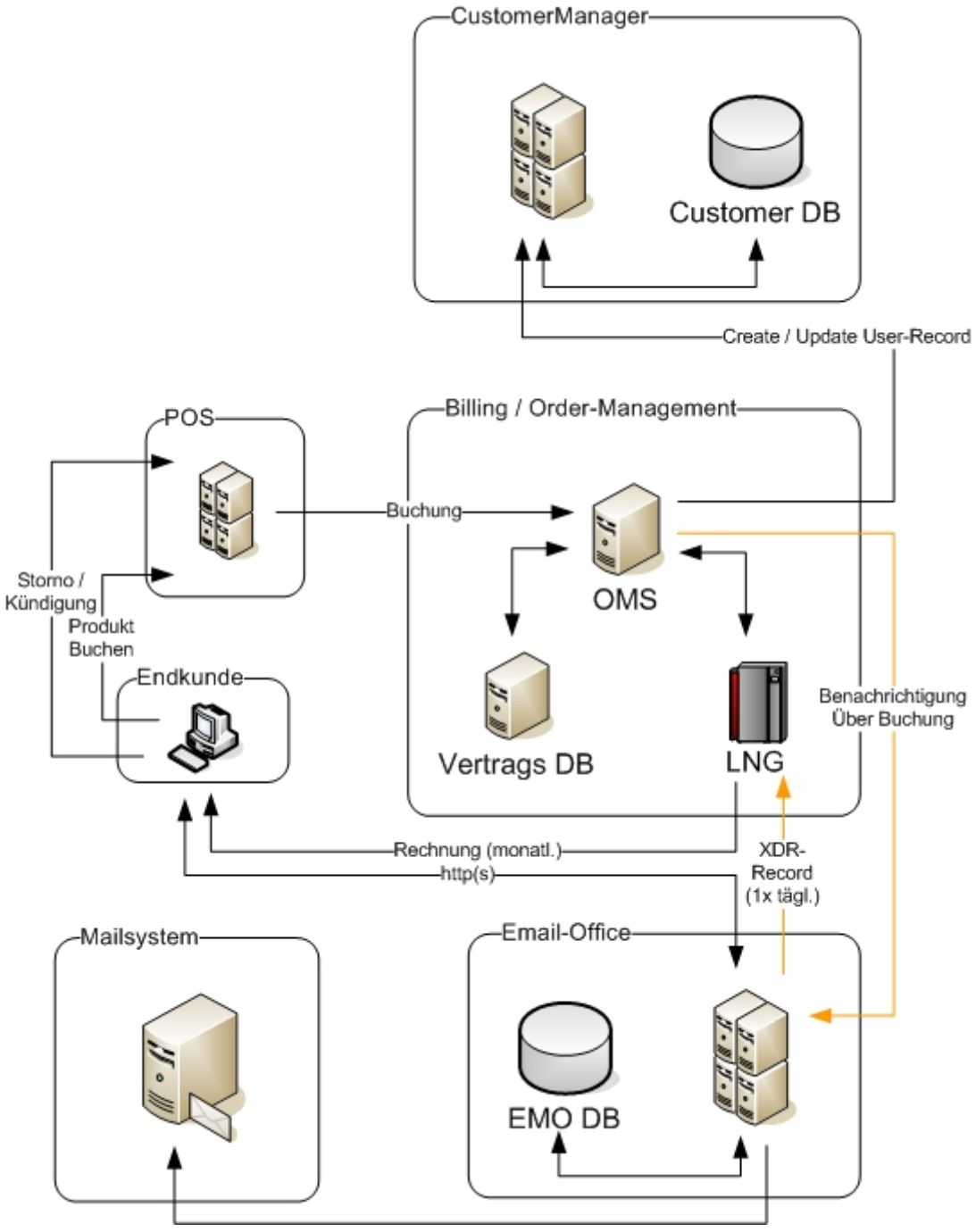

Abbildung 2.3.: Systemlandschaft einer Bezahlbeziehung

3. Lösungsansatz der Nutzungsdatenverwaltung

Das Kapitel 2.2 hat die aktuelle Systemlandschaft untersucht und gezeigt, dass im heutigen Zustand jede Applikation die Nutzungsdatenverwaltung für sich vornehmen muss. Es wurde ausserdem angedeutet, dass dieses Verhalten Gefahren birgt, wenn mehrere Applikationen mit den gleichen Nutzungsdaten arbeiten. Ganz zu schweigen davon, dass innerhalb dieser Systemlandschaft jede Applikation die Nutzungsdatenverwaltung selber implementieren muss, was eine Verschwendung von Ressourcen ist und zusätzliche Entwicklungszeit bedeutet.

Die Gefahren und Probleme, die eine applikationseigene Nutzungsdatenverwaltung birgt, sind:

- Synchronisation:
  Die Applikationen müssen sich untereinander in Echtzeit synchronisieren, um eine Überlieferung[1] zu vermeiden.
- Ressourcen:
  Die applikationseigene Implementierung der Nutzungsdatenverwaltung führt zu einer Ressourcenverschwendung.
- Entwicklungszeit:
  Die Entwicklung neuer Anwendungen, die eine Nutzungsdatenverwaltung benötigen, dauert länger.

[1] Überlieferung: Wenn der Kunde mehr Verbraucht, als ihm zusteht.

- Fehlerquellen:
  Jede weitere unnötige Implementierung einer bestimmten Komponente stellt eine weitere Fehlerquelle dar, die vermieden werden könnte.

Fazit dieser Tatsachen ist, dass die Nutzungsdatenverwaltung aus den Applikationen herausgezogen und als eigenständiger zentraler Service zur Verfügung gestellt werden muss, welches das Ziel dieser Diplomarbeit darstellt.

3.1. Integration ins Billingsystem

Die offensichtliche Lösung ist, dass die Nutzungsdaten direkt aus dem Billingsystem heraus genutzt werden. D.h. es wird eine neue Schnittstelle zu dem Billingsystem geschaffen, damit die Applikationen das Billingsystem als Nutzungsdatenverwaltungssystem benutzen können.

Die offensichtliche Lösung ist leider nicht immer die beste und bei genauerer Betrachtung stellt man fest, dass dieser Ansatz sogar ganz und gar unbrauchbar ist.

Kundengruppen

Grundsätzlich können die Kunden zwei verschiedenen Gruppen zugeordnet werden. Zum einen gibt es die Paid-Kunden und zum anderen gibt es die so genannten Basic-Kunden. Paid-Kunden sind solche Kunden, mit denen eine Bezahlbeziehung existiert, wohingegen Basic-Kunden alle diejenigen Kunden sind, die noch kein Produkt erworben haben.

Dem Billingsystem sind *nur* Paid-Kunden bekannt und aus diesem Grund ist die Existenz von Basic-Kunden alleine schon ein Ausschlusskriterium.

Obwohl die Basic-Kunden für einen Service nicht bezahlt haben, ist es unter Umständen trotzdem, z.B. zu Werbezwecken, gewünscht, diesen Kunden einen

Service in gemindertem Mass zur Verfügung zu stellen. Am Beispiel des SMS-Nachrichten Versandes erhalten die Basic-Kunden monatlich die Möglichkeit 3 SMS-Nachrichten zu verschicken.

Weil die Basic-Kunden dem Billingsystem nicht bekannt sind, kann das Billingsystem keine Nutzungsdatenverwaltung für diese Kundengruppe vornehmen.

Leistung

Das rating und die Rechnungsstellung sind sehr komplexe und zeitaufwendige Vorgänge, die täglich einen Grossteil der Systemressourcen des Billingsystems in Anspruch nehmen. Die Nutzungsdatenverwaltung würde bedeuten, dass zusätzlich mehrere hundert oder sogar tausend Anfragen pro Sekunde an das Billingsystem gestellt werden. Im besten Fall würde das zu langen Wartezeiten führen und im schlimmsten Fall dazu, dass die tägliche Rechnungsstellung nicht mehr fertig gestellt werden kann.

Verfügbarkeit

Es kann unter Umständen gewünscht sein, das Billingsystem Offline zu nehmen. Sei es für Wartungsarbeiten oder, um ein Update für eines der diversen Komponenten des Billingsystems zu installieren. Falls die Nutzungsdatenverwaltung ein Bestandteil des Billingsystems wäre, wäre dies nur noch beschränkt möglich, ohne eine Beeinträchtigung der Kunden nach sich zu ziehen.

Ausfallsicherheit

Leider ist es eine Tatsache, dass das Billingsystem und / oder der Datenbankserver des Billingsystems instabil sind und es fast wöchendlich zu “kurzzeitigen” Ausfällen kommt. Dieses ist für die Nutzungsdatenverwaltung inakzeptabel, weil die Kunden direkt dadurch betroffen sind.

Somit ist die Integration der Nutzungsdatenverwaltung in das Billingsystem nicht nur nicht wünschenswert, es ist sogar unmöglich.

3.2. Eine eigenständige Nutzungsdatenverwaltung

Im Kapitel 3.1 wurde gezeigt, dass das Billingsystem für die Nutzungsdatenverwaltung nicht in Frage kommt und ein weiteres zentrales System, welches diese Funktionalität zur Verfügung stellen könnte, existiert (noch) nicht. Die Schlussfolgerung dieser Erkenntnis ist, dass für die Integration von Nutzungsdaten heterogener Dienstplattformen ein neuen Server entwickelt werden muss.

Wenn man die in Abbildung 2.3 dargestellten Systemlandschaft mit dem Hintergedanken einer zentralen Nutzungsdatenverwaltung betrachtet, müsste diese zwischen dem Billing / Order-Management und der Applikation angeordnet werden.

Abbildung 3.1 zeigt, wie die Systemlandschaft aussehen wird, nachdem der neue zentrale Dienst für die Nutzungsdatenverwaltung eingeführt wurde.

Wie im aktuellen Prozess, wird bei einer Bestellung eine Benachrichtigung über die Buchung von dem Order Management System geleistet. Diese Benachrichtigung geht nun anstatt an die Applikation an die Nutzungsdatenverwaltung. Die Nutzungsdatenverwaltung legt für den Kunden ein neues Konto an, wenn nicht schon eines vorhanden ist, und lädt dieses Konto auf.

Die verschiedenen Applikationen können den aktuellen Kontostand bei der Nutzungsdatenverwaltung anfragen und, wenn das Konto ausreichend gedeckt ist, diese Konten belasten. Die Nutzungsdatenverwaltung wiederum übernimmt die Aufgabe, die Verbrauchsdaten an das Billingsystem zu melden, damit der Verbrauch bepreist und verbucht werden kann.

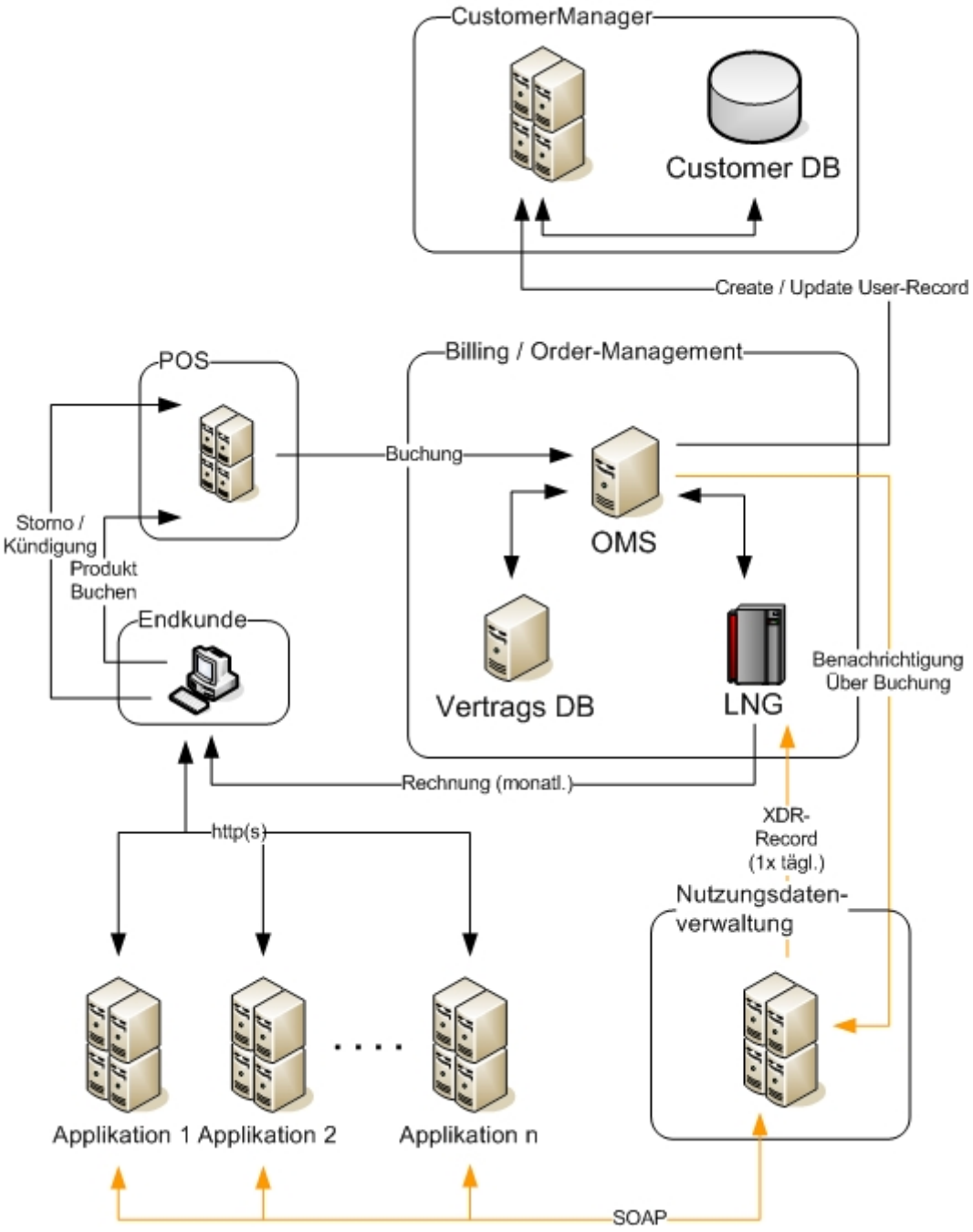

Abbildung 3.1.: Systemlandschaft mit zentraler Nutzungsdatenverwaltung

3.3. Allgemeine Betrachtung

Wenn die Prozesse und Komponenten generalisiert werden, kann die gesammte Systemlandschaft, unabhängig von der freenet.de AG, generell aus vier Bausteinen zusammengestellt werden. Im Centrum steht die Nutzungsdatenverwaltung, für die es einen Zulieferer (Bestellung), einen Konsumenten (Anwendung) und einen Empfänger (Rechnungsstellung) gibt (siehe Abbildung 3.2). Unabhängig davon, wo und in welcher Systemumgebung die Nutzungsdatenverwaltung eingesetzt wird, wird es auf die eine oder andere Art und Weise immer einen Zulieferer, Konsumenten und Empfänger geben.

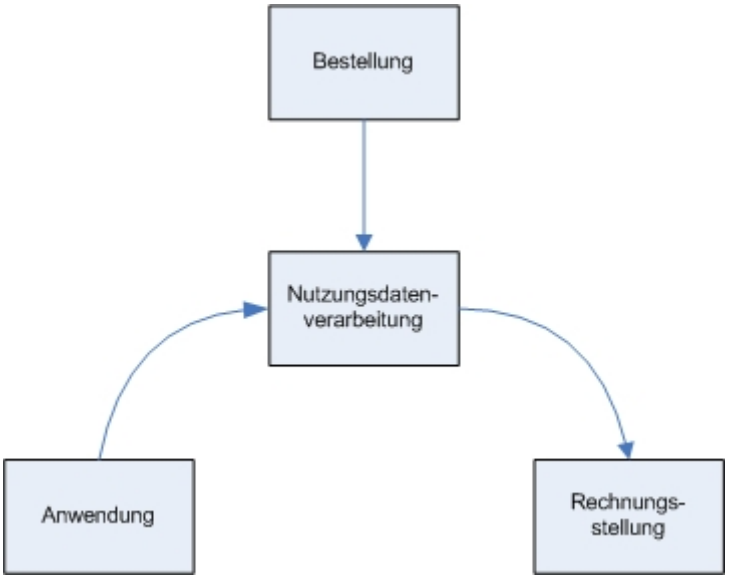

Abbildung 3.2.: Generalisierte darstellung des Prozesses

Die Nutzungsdatenverwaltung wird so offen und generell einsetzbar wie möglich realisiert, damit man nicht auf bestimmte Applikationen und Typen festgesetzt ist. Um dieses zu erreichen, wird die Definition, der zu verwaltenden Kontingente, aus der Nutzungsdatenverwaltung herausgezogen und dem Zulieferer bzw. dem Konsumenten überlassen. D.h., die Nutzungsdatenverwaltung übernimmt zwar die Verwaltung der Kontingente aber die Definition dieser Kontingente, die verwaltet werden, wird den benutzenden Systemen überlassen. Durch diesen Ansatz wird

es ermöglicht, dass neue Applikationen sofort und ohne zusätzlichen Aufwand die Nutzungsdatenverwaltung einsetzen können. Wenn zwei Applikationen die gleiche Definition für die Kontingente verwenden, werden von der Nutzungsdatenverwaltung auch die gleichen Konten verwendet.

Innerhalb der freenet.de AG Systemlandschaft wird dieses erreicht, indem die Definition der Kontingente dem Produkt Katalog hinzugefügt wird. Wenn ein neues Produkt verkauft werden soll, müssen lediglich im Produktkatalog die Einzelheiten über die Definition der Kontingente und deren Abrechnung hinterlegt werden und die Nutzungsdatenverwaltung übernimmt diese werte.

4. Randbedingungen

Randbedingungen sind Vorgaben von der freenet.de AG, die bei der Erfüllung des Zieles dieser Arbeit einzuhalten sind. Dabei handelt es sich auf der einen Seite darum, dass Systeme, die bei der freenet.de AG bereits existieren, auch weiterhin zum Einsatz kommen und, dass diese nicht durch eigene Komponenten / Prozesse innerhalb der Nutzungsdatenverwaltung ersetzt werden. Zum anderen geht es um den Einsatz von bestimmten Tools und Techniken, mit denen die Firma gute Erfahrung gemacht hat und wo das nötige Know-how für die Weiterentwicklung der Nutzungsdatenverwaltung bzw. für die Wartung des selbigen auch nach der Beendigung dieser Arbeit existiert.

- Vorhandene Systeme
  - Customer-Manager (Anhang A.2)
  - Produkt-Katalog (Anhang A.4)
  - Customer-Information-Service
  - Vertrags Datenbank
- Technik und Tools
  - SOAP (Axis) (Anhang B.1)
  - Persistenz Schicht (Cayenne) (Anhang B.2)

4.1. Vorhandene Systeme

4.1.1. Customer-Manager

Bei dem Customer-Manager handelt es sich um eine Sybase-Serverfarm[1], welche die Kundenstammdaten aller Kunden der freenet.de AG enthält und verwaltet. Für den Zugriff auf die Sybase-Serverfarm gibt es wiederum eine SOAP-Serverfarm, welche Methoden für den Zugriff auf die Kundenstammdaten zur verfügung stellen.

Die von dem Customer-Manager verwalteten Kundenstammdaten bestehen in erster Linie aus den Standart Stammdaten wie Name, Adresse, Kontakt usw., sowie aus den Payment Informationen. Die CID, also die Customer ID, welche als Identifier innerhalb des Customer-Managers zur Verwendung kommt, stellt den freenet weiten Identifier eines jeden Kunden dar.

Weitere Einzelheiten über den Customer-Manager finden sich im Anhang A.2.

4.1.2. Produkt-Katalog

Der Produkt-Katalog verwaltet Produktdefinitionen für alle Produkte, die die freenet.de AG ihren Kunden anbietet. Indem die Produktverwaltung an einem zentralen Service gebunden ist, wird vermieden, dass Inkonsistenzen zwischen den verschiedenen Systemen entstehen können. Lediglich durch Angabe der Produkt ID kann jedes System die für seine Zwecke nötigen weiteren Informationen wie z.B. Preis, Name, Rechnungstext usw. aus dem Produkt-Katalog anfordern.

Der Produkte-Katalog verwaltet ...

- Produktnamen & -nummern
  - für verschiedene Sichten

[1] `http://www.sybase.com/`

- Gültigkeiten & Versionen
  - für verschiedene Sichten
- Eigenschaften (Texte, Parameter, ...)
  - für verschiedene Sichten
- Abhängigkeiten
  - z.B. "Produkt A immer nur im Zusammenhang mit Produkt B aber nie zusammen mit Produkt C"

Weitere Einzelheiten über den Produkt-Katalog befinden sich im Anhang A.4.

4.1.3. Customer-Information-Service

Der Customer-Information-Service (kurz CIS) ist die Primäre Informationsquelle, die vom Freenet Customer Care benutzt wird, um Kundenanfragen zu beantworten sowie auch allgemein für administrative Zwecke von Kundenangelegenheiten. Hierbei handelt es sich lediglich um eine Web basiertes Beauskunftungssystem, dass mit Hilfe von SOAP Zugriffen auf die diversen Systeme eine zentrale, einheitliche und intuitive Oberfläche zur Kundenbetreuung zur Verfügung stellt.

Um dieses Konzept nicht zu sprengen, ist es nötig, dass das administrative Frontend für die Nutzungsdatenverwaltung innerhalb dieses Customer Information Services integriert wird. Die Nutzungsdatenverwaltung wird zu diesem Zwecke administrative Methoden zur Verfügung stellen, mit denen die Zustände der Kunden im Rahmen der Nutzungsdatenverwaltung abgefragt und geändert werden können.

4.1.4. Vertrags Datenbank

Die Vertrags DB steht im Gegensatz zu den anderen Systemen nur indirekt mit der Nutzungsdatenverwaltung im Zusammenhang. Die Vertrags DB ist das über-

geordnete System, welches die Vertragsinformationen jeglicher Verträge zwischen der freenet.de AG und ihren Kunden hält.

Die Nutzungsdatenverwaltung benötigt jederzeit die Informationen über die aktiven Verträge, um z.B. monatliche Abo Produkte zu verarbeiten und die Konten der Kunden am Ende des Abrechnungszeitraumes wieder aufzuladen. Aus diesem Grund werden parallel zur Vertrags DB, die Informationen über aktive Verträge auch innerhalb der Nutzungsdatenverwaltung gehalten. Um ein eventuelles Auseinanderlaufen der beiden Systeme zu verhindern, ist es nötig, eine Schnittstelle zu haben, mit deren Hilfe der aktuelle Vertragsstand der beiden Systeme verglichen und Abweichungen gemeldet werden können.

4.2. Technik und Tools

4.2.1. WebService - SOAP (Axis)

Seit der Entwicklung der SOAP Spezifikation wurden diverse Implementierungen der selbigen in allen gängigen Programmiersprachen entwickelt. Obwohl SOAP in Hinblick auf Overhead und Geschwindigkeit klar im Nachteil ist zu anderen Schnittstellen wie z.B. RMI, hat sich die freenet.de AG aufgrund von Einheitlichkeit und Wiederverwendbarkeit dafür entschieden, SOAP als primäre Kommunikationsschnittstelle in allen ihren System einzusetzen.

Nachdem die gängigsten Java-SOAP Implementierungen getestet und evaluiert wurden, wurde die unternehmensweite Entscheidung getroffen, dass zur Schnittstellenentwicklung unter Java die Implementierung der SOAP Spezifikation von Apache.org - Axis eingesetzt werden soll.

Weitere Einzelheiten über die SOAP Implemenatierung Axis befinden sich im Anhang B.1.

4.2.2. Persistenz Schicht (Cayenne)

Cayenne ist eine Open Source Java Implementierung einer Persistenz Schicht für Objekt / Relationales mapping. Ein Objekt / Relational mapping ist eine Darstellung von Tabellen und Schemen einer relationalen Datenbank als Objekte innerhalb einer Objekt Orientierten Programmiersprache, in diesem Fall Java. Durch den intelligenten Einsatz eines Objekt / Relationalen Frameworks ist es möglich, die Entwicklungszeit eines Datenbankprojektes stark zu senken, da innerhalb des Projektes nur noch mit Objekten gehandelt wird und die eigentlichen SQL Statements automatisch im Hintergrund erledigt werden.

Die Hauptmerkmale von Cayenne sind:

- Verwaltung von Persistenten Java Objekten (Abbildungen von Relationalen Datenbanken)
- Einfaches Erstellen von Queries und Updates
- Atomic Updates von allen geänderten Objekten
- Transaktionsfähigkeit unabhängig von der Datenbank
- Verschmelzung von mehreren Datenbanken in eine virtuelle Datenquelle
- GUI Oberfläche als Modeller für
  - Reverse-Engineering von RDBMS Schemen
  - Arbeiten mit Datenbank mappings
  - Automatische Generierung von Java Sourcecode der Persistenten Objekte

Cayenne wurde erfolgreich in mehreren Projekten der freenet.de AG eingesetzt und soll auch weiterhin zum Einsatz kommen.

Weitere Einzelheiten zu Cayenne befinden sich im Anhang B.2.

5. Architektur und Design

Bevor die Architektur und das Design der Nutzungsdatenverwaltung beschrieben wird, soll zuerst eine kurze Zusammenfassung erstellt werden.

Das Ziel der Nutzungsdatenverwaltung heterogener Dienstplattformen ist die Bereitstellung eines zentralen Dienstes, welcher unabhängig von der Art des Dienstes eine Verwaltung von verschiedenen Kontingenten und Verbrauchsdaten auf eine einheitliche Art und Weise zur Verfügung stellt.

Somit soll erreicht werden, dass

1. schnell und problemlos neue Anwendungen, die eine Nutzungsdatenverwaltung benötigen, entwickelt werden können.
2. anwendungsübergreifende Nutzungsdaten zentral Verwaltet werden, um Konflikten und Inkonsistenten Daten vorzubeugen.
3. Anwendungen sich auf das wesentliche (die Bereitstellung des Dienstes) beschränken.
4. Anzahl und Diversität der Fehlerquellen aufgrund der Nutzungsdatenverwaltung beschränkt werden.

5.1. Architektur

Als Architekturmodell der Nutzungsdatenverwaltung wird die 3-Tier-Architektur eingesetzt. Die Grundlage jeder 3-Tier-Architektur ist die Aufteilung der Applika-

tions-Benutzeroberfläche (Presentation), der Anwendungs-Logik (Business objects) und der Datenbank (relational Database).

Durch die Anwendung dieses Teile-und-Herrsche-Entwicklungsansatzes werden folgende Vorteile erhofft:

- Bessere Skalierbarkeit:
  Dadurch, dass die Architektur in die verschiedenen Schichten gespalten wird, ist es möglich, eine komplette Schicht auf einen anderen Server oder sogar auf eine ganze Serverfarm auszulagern.
- Wartbarkeit:
  Die Schichten können unabhängig voneinander gewartet und sogar ganz ersetzt werden.
- Gute Code Wiederverwendung:
  Teile des Codes könne sehr einfach angepasst und auch in anderen Services oder Applikationen eingesetzt werden.
- Hohe Entwicklungsproduktivität:
  Durch die Aufspaltung des Prozesses in kleinere Teilaufgaben ist die Entwicklung und auch die Wartung der einzelnen Bestandteile sehr viel einfacher.

Abbildung 5.1 stellte die 3-Tier-Architektur, die bei der Nutzungsdatenverwaltung zum Einsatz kommt, dar. Die relationale Datenbank ist an und für sich keine eigene Schicht. Stattdessen erzeugt sie mit der Persistenzschicht zusammen die Datenbankschicht der 3-Tier-Architektur. Um die Vorgänge besser untersuchen zu können, werden aber im folgenden die beiden als eigenständige Schicht behandelt.

5.2. Entwurf

Aufgrund der im Kapitel 4 genannten Randbedingungen, die Eingehalten werden müssen, sind die Auswahlmöglichkeiten, die bei dem Entwurf der Nutzungsdaten-

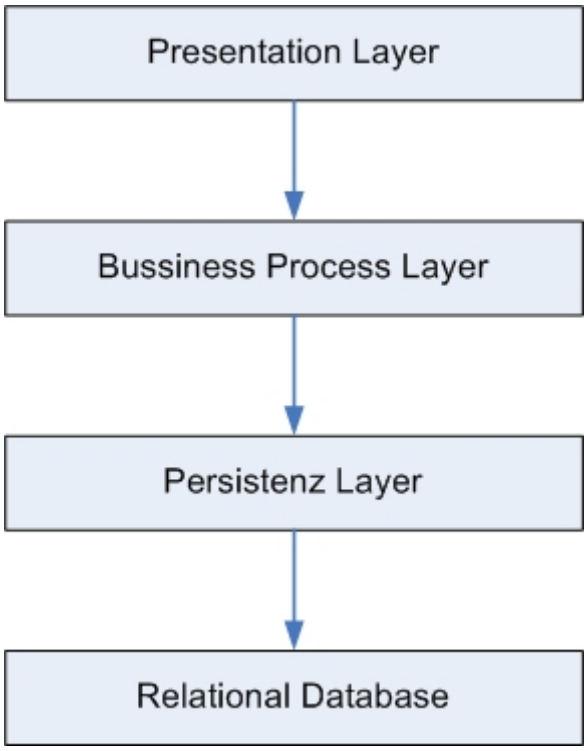

Abbildung 5.1.: Darstellung der 3-Tier-Architektur

verwaltung gewählt werden können, etwas eingeschränkt. Die eingesetzte Präsentationsschicht wird eine Axis SOAP Schnittstelle werden und als Persistenzschicht wird Cayenne eingesetzt werden.

Abbildung 5.2 zeigt den Entwurf mit drei voneinander völlig unabhängigen Applikationsservern, die sich untereinander abstimmen und für die konsistenz der Daten sorgen. Wie viele Applikationsserver im einzelnen zum Einsatz kommen und wie die Konfiguration im Detail aussehen wird, ist aufgrund der verwendeten Architektur nicht relevant und kann später in Abhängigkeit der an das System gestellten Anforderungen beliebig angepasst werden. So wäre es zum Beispiel denkbar, dass nur eine Datenbank existiert und mehrere Server, auf denen die Business Logik implementiert ist.

In diesem Szenario wird davon ausgegangen, dass drei Datenbanken existieren,

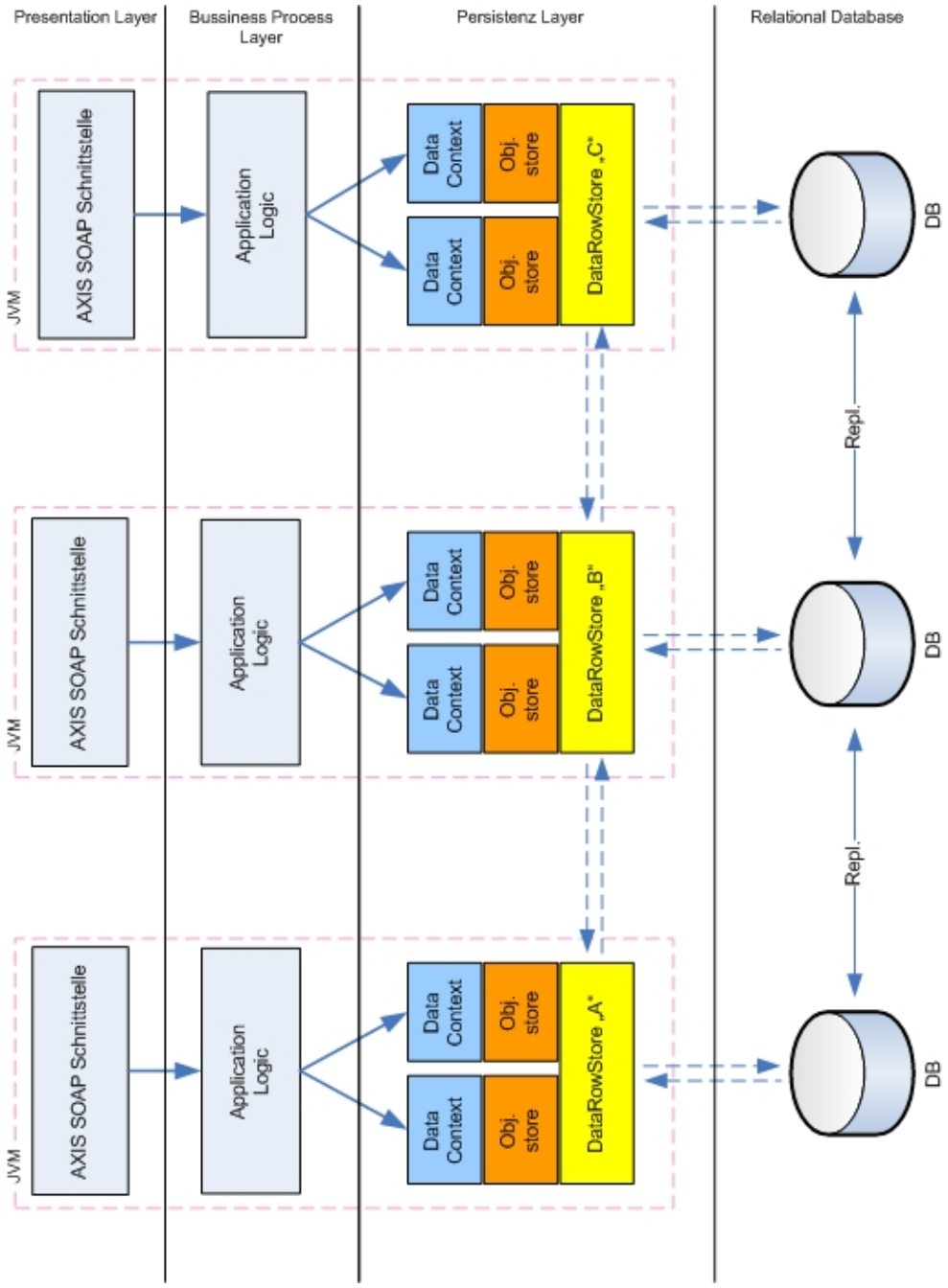

Abbildung 5.2.: Entwurf der 3-Tier-Architektur

die sich untereinander replizieren. Nur in den wenigsten Fällen ist es zwingend notwendig, dass mehr als eine Datenbank zum Einsatz kommt. In den meisten Anwendungen ist eine gut strukturierte Datenbank völlig ausreichend, um alle Anfragen zufriedenstellend zu beantworten. Jedoch ist es bei wichtigen Prozessen empfehlenswert, zumindest eine Standby-Datenbank zu besitzen, für den Fall, dass die Datenbank oder der Server Fehler verursachen.

Als Persistenzschicht wird, wie bereits erwähnt, Cayenne eingesetzt. Cayenne hat einige sehr hilfreiche Features, die genutzt werden. An dieser Stelle ist es notwendig zuerst einmal die grundlegende Arbeitsweise von Cayenne zu Verstehen.

Der erste Schritt, wenn mit Cayenne gearbeiten wird, ist, dass ein DataContext erzeugt wird. Wenn Cayenne mit der Welt von Datenbanken verglichen wird, ist ein DataContext ungefähr das gleiche, wie eine Transaktion. Die Informationen, die in der Datenbank stehen, werden bei Bedarf in den DataRowStore eingelesen. Wenn die gleichen Informationen ein weiteres mal benötigt werden oder wenn z.B. mehrere DataContexte mit den gleichen Daten arbeiten wollen, werden die Daten im DataRowStore benutzt, anstatt jedesmal aus der Datenbank ausgelesen zu werden. Der DataRowStore ist auch dafür zuständig, Änderungen, die von einem DataContext an den Daten vorgenommen werden, durch Snapshot Events an alle Registrierten DataContexte weiterzugeben sowie die Änderungen an die Datenbank zu propagieren.

Ein DataContext liest die Daten, mit denen es Arbeiten möcht, in den Object Store. Im Object Store werden auf Grundlage der vorliegenden Daten komplette Java Objekte gezeugt und gehalten. Änderungen an einzelnen Objekten werden nicht sofort an das DataRowStore weitergeleitet. Erst mit einem commit auf dem DataContext werden alle Änderungen, die an Objekten dieses Object Stores vorgenommen wurden, an das DataRowStore und somit an die Datenbank weitergegeben. Dieser Vorgang wird als ein Atomic Update realisiert, so dass andere DataContexte in der Zeit in der die Updates durchgeführt werden, keine eigenen Änderungen vornehmen oder Daten auslesen können.

Durch den Einsatz von Snapshot Events wird gewährleistet, dass innerhalb einer

Java Virtuellen Maschine Änderungen am Datenbestand durch einen DataContext sofort an alle weiteren DataContexte weitergeleitet werden, welche entsprechend auf diese Änderungen reagieren. Ferner wurde ein Protokoll implementiert, welches es erlaubt, dieses Verhalten sogar über die Grenzen einer Java Virtuellen Maschine hinaus zu tragen. Es ist dadurch möglich, mehrere DataRowStores zu einer art Cluster zusammenzuschalten, so dass Snapshot Events nicht nur an die DataContexte innerhalb einer Java Virtuellen Maschine sondern an alle anderen DataRowStores innerhalb dieses Clusters weitergeleitet werden.

In dem vorliegenden Entwurf mit drei Applikationsservern läuft auf jedem Server eine eigene virtuelle Maschine. Die drei DataRowStores wurden zu einem Cluster zusammengeschaltet und propagieren die Snapshot Events untereinander. Mit diesem Konstrukt ist es möglich, sehr schnell und vor allem auch sehr einfach auf Lastprobleme zu reagieren, ohne das auf dem Business Process Layer dafür sorgen getragen werden muss, dass keine Inkonsistenzen entstehen.

Der Business Process Layer enthält die eigentliche Applikationslogik. In dieser Schicht werden die Daten verarbeitet, Berechnungen gemacht, Entscheidungen getroffen usw.

Letztendlich gibt es noch die Persistenzschicht. Hierbei handelt es sich lediglich um einen Axis WebService, der die Applikations Logik, die im Business Process Layer bereitgestellt wurde, als SOAP WebService verpackt, anderen Applikationen zur Verfügung stellt.

Innerhalb dieses Entwurfes ist es möglich, jede Schicht für sich unabhängig von den anderen Schichten beliebig zu skalieren, auszuwechseln oder zu erweitern. Z.B. könnte es in Zukunft durchaus sinnvoll werden die Präsentationsschicht zu erweitern und neben einer SOAP Schnittstelle auch weitere Schnittstellen wie z.B. RMI zur Verfügung zu stellen. Durch den Einsatz von SOAP wurde bereits gewährleistet, dass die Schnittstelle zumindest in allen gängigen Programmiersprachen angesprochen und benutzt werden kann. Nichts desto trotz hat auch SOAP gewisse Nachteile. Beim Einsatz von SOAP ist die Menge der zu übertragenden Daten erheblich höher (ca. Faktor 25 beim Request und Faktor ¿500 beim Response) ver-

glichen mit anderen, einfacheren RPC-Protokollen. Dies ist nicht nur ein Problem der beanspruchten Netzwerkbandbreite, auch das Generieren und Auswerten (Parsen) der Nachrichten erfordert sehr viel mehr Rechenzeit. Es ist in dieser Phase des Projektes jedoch nicht abzuschätzen, ob daraus die Notwendigkeit hervorgeht, für bestimmte Prozesse und Applikationen eine andere Schnittstelle als SOAP zur verfügung zu stellen.

5.3. Prozesse der Nutzungsdatenverwaltung

Zur Vervollständigung des Designs ist es ausserdem noch nötig, die Prozessabläufe zu entwerfen, damit die Implementierung dementsprechend gestaltet werden kann, um diese Prozessabläufe zu gestatten.

5.3.1. Bestandsabfrage

Eine Bestandsabfrage wird überwiegend in zwei Fällen benötigt. Erstens muss eine Applikation, bevor sie es einem Kunden überhaupt ermöglicht den Versuch zu unternehmen, einen Dienst in Anspruch zu nehmen, überprüfen, ob der Kontostand dieses erlaubt. Zweitens möchte eine Applikation den Kunden über seinen aktuellen Kontostand informieren, damit der Kunde sich dementsprechend verhalten und eventuell einplanen kann, sein Konto wieder aufzuladen. Ein Sonderfall kann entstehen, wenn man dem Kunden in bestimmten Fällen erlauben möchte seinen Kontostand zu überziehen. Als Beispiel sollen wieder die SMS-Pakete dienen. Der Kunde erwirbt z.B. das Recht 50 SMS-Nachrichten zu verschicken. Anstatt es dem Kunden zu verbieten weitere SMS-Nachrichten zu verschicken, nachdem die 50 SMS-Nachrichten verbraucht sind, erlaubt man es ihm, sein Konto zu überziehen und stellt dem Kunden jede weitere SMS-Nachrichten gesondert (meist zu einem etwas höheren Preis) in Rechnung. In diesem Fall ist der Kontostand zwar leer, trotzdem darf der Kunde den Dienst in Anspruch nehmen. Somit wird eine Überprüfung benötigt, die in Abhängigkeit der Produktdaten sicherstellt, dass ein

Kunde einen Dienst beanspruchen darf oder nicht. Diese beiden Prozessabläufe sind in Abbildung 5.3 dargestellt.

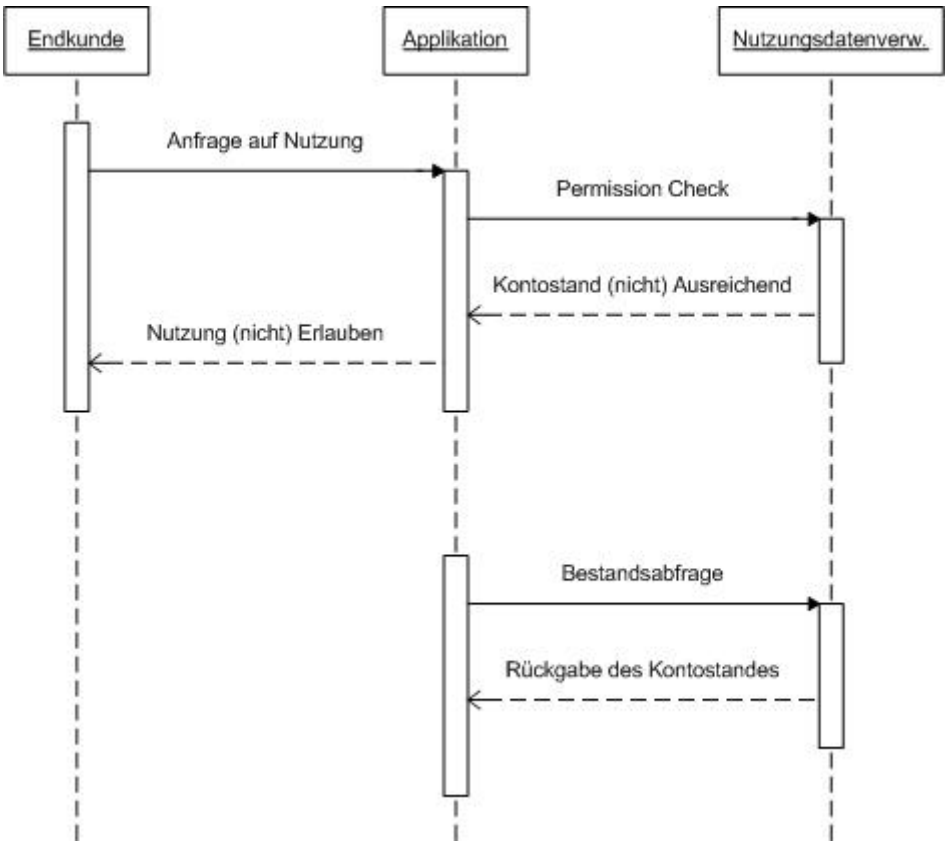

Abbildung 5.3.: Prozessablauf der Bestandsabfrage und der Rechte-Überprüfung

5.3.2. Buchung

Nicht Erfolgreich

Prinzipiell ist dieser Prozessablauf dem vorherigen sehr ähnlich. Die initiale Rechte-Überprüfung war erfolgreich und der Kunde möchte nun den Dienst in Anspruch nehmen und zum Beispiel seine SMS-Nachrichten verschicken. Leider hat sich aber

in der Zwischenzeit etwas am Kontostand des Kunden geändert, weil er zum Beispiel in einem zweiten Fenster schon eine andere SMS-Nachrichten verschickt hat, wodurch sein Kontostand nicht mehr ausreichend ist. Die Ausführung der angebotenen Dienstleistung muss somit abgelehnt werden, wie es in Abbildung 5.4 dargestellt ist.

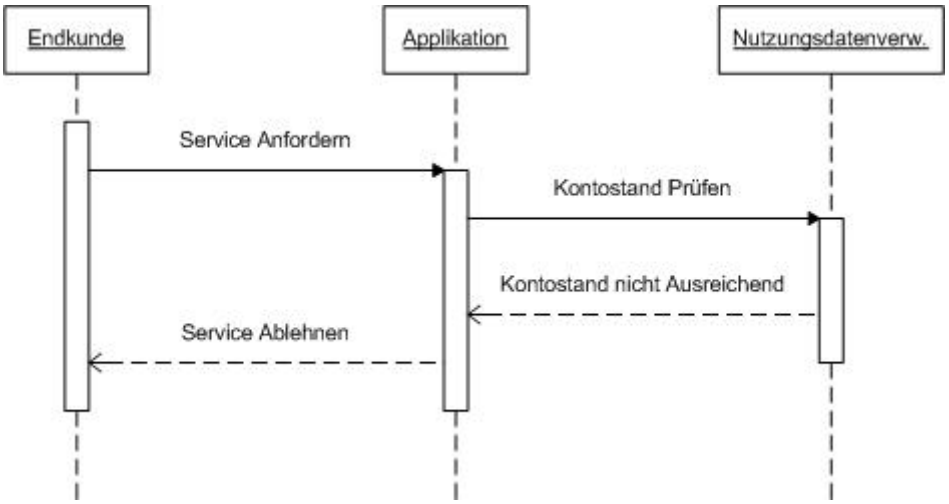

Abbildung 5.4.: Prozessablauf einer Ablehnung der Ausführung

Teilweise Erfolgreich

Untersucht wird nun der Prozess, wenn der Kunde einen Dienst anfordert und der Kontostand des Kunden ausreichend ist. Die Auslieferung des Dienstes an den Kunden durch die Applikation kann sich unter Umständen über einen längeren Zeitraum hinziehen und es ist auch nicht sichergestellt, dass die Auslieferung tatsächlich stattfindet. Am Beispiel des Downloads eines Musikstückes oder einer Software kann dies am besten verdeutlicht werden. Die Applikation muss dem Kunden die Möglichkeit des Downloads bieten. In diesem Augenblick muss der Kontostand dieses wiederspiegeln, weil der Kunde jederzeit den Download starten kann. Ansonsten wäre es möglich, X beliebige Musikstücke auszuwählen, obwohl laut Kontostand nur eines erlaubt ist, und den Download aller Musikstücke parallel zu starten. Allerdings ist es nicht sicher, dass der Kunde auch tatsächlich den

Download überhaupt tätigt. In diesem Fall darf dem Kunden ein nicht in Anspruch genommener Service auch nicht in Rechnung gestellt werden.

Somit muss das Vorgehen folgendermassen sein: die Applikation muss bei Verfügbarkeit des Dienstes eine Reservierung durchführen, um sicher zu stellen, dass das Konto nicht überzogen wird. Nachdem sichergestellt wurde, dass der Dienst auch in Anspruch genommen wurde, muss aus der Reservierung eine Buchung gemacht werden. Erst bei einer Buchung wird der Wert tatsächlich vom Konto abgebucht. Wenn allerdings die Buchung nie getätigt wird, muss nach einer angemessenen Zeit die Reservierung verfallen, damit der Kunde diese zuvor reservierten Werte anderweitig benutzen kann. Dieses Verhalten ist in Abbildung 5.5 dargestellt.

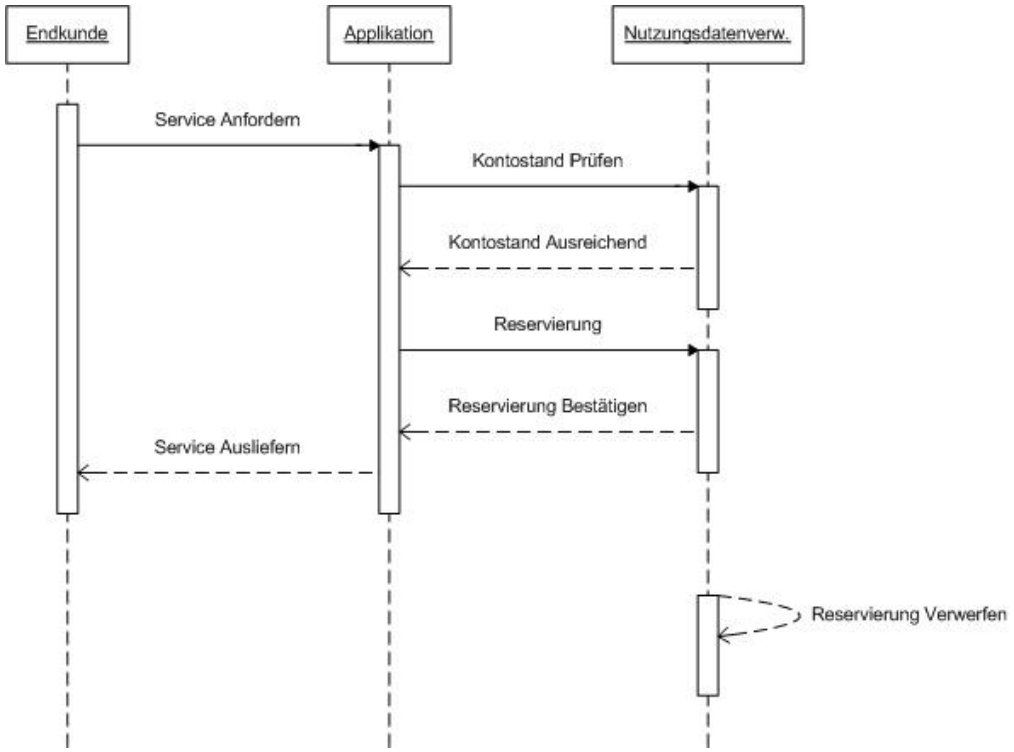

Abbildung 5.5.: Prozessablauf einer Reservierung ohne Buchung

Erfolgreich

Parallel zum vorherigen Prozessablauf muss die Applikation entscheiden, zu welchem Zeitpunkt eine Auslieferung einer Dienstleistung als erfüllt betrachtet wird. Wenn dieser Zeitpunkt erreicht ist, muss die Applikation aus einer Reservierung eine Buchung machen. Damit die Applikation bestimmen kann, welche Reservierung gebucht wird, liefert die Nutzungsdatenverwaltung bei einer Reservierung einen Identifier, den sich die Applikation merken muss. Unter Verwendung dieses Identifiers und des Kunden Identifiers kann danach eine Buchung getätigt werden.

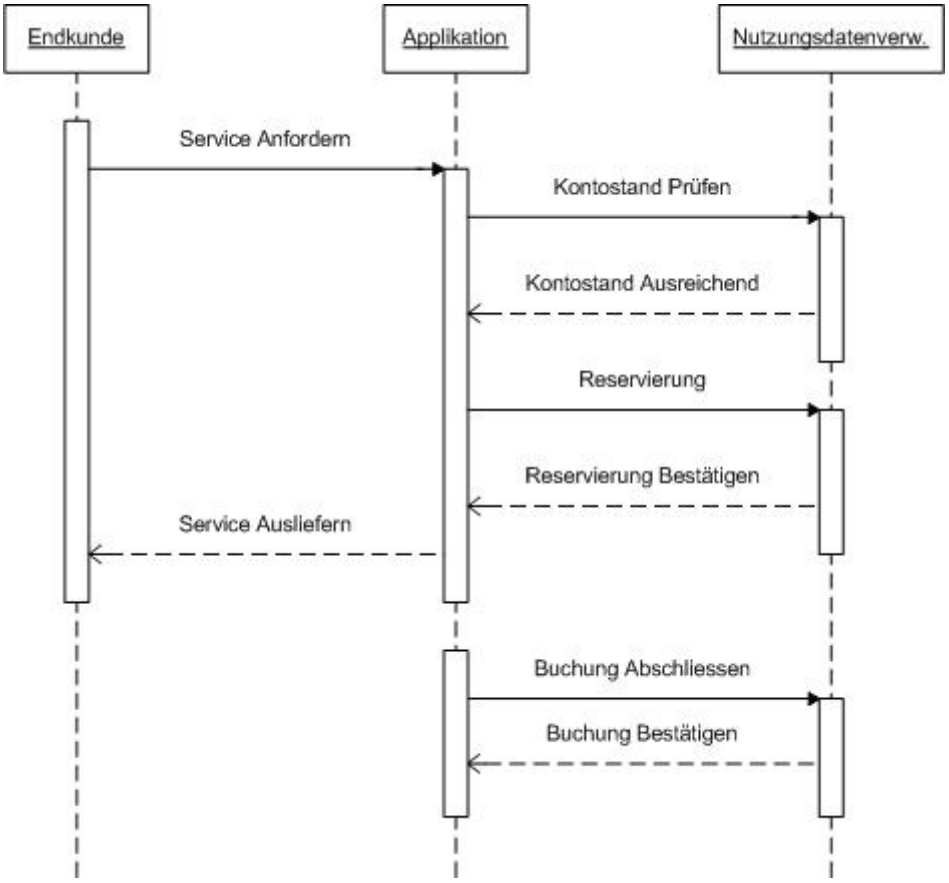

Abbildung 5.6.: Prozessablauf einer Reservierung und Anschliessender Buchung

Bei der Buchung werden beide Identifier benötigt um einen eventuellen Missbrauch durch Kunden oder fehler der Applikationen auszuschliessen. Nur wenn die Reservierung dem richtigen Kunden zugeordnet ist wird die Buchung angenommen.

5.4. Web Service

Ein Web Service ist eine Software-Anwendung, die mit einem Uniform Resource Identifier (URI) eindeutig identifizierbar ist und deren Schnittstellen als XML-Artefakte definiert, beschrieben und gefunden werden können. Ein Web Service unterstützt die direkte Interaktion mit anderen Software-Agenten unter Verwendung XML-basierter Nachrichten durch den Austausch über internetbasierte Protokolle. Web Services spielen als Middleware im Bereich E-Business eine zunehmend bedeutende Rolle.

5.4.1. Architektur

Client-Programme senden im Allgemeinen Anfragen an einen Web Service und dieser antwortet mit der gewünschten Information. Von vielen Seiten wird daher behauptet, dass Web Services für Rechner das sind, was Webseiten für den Menschen sind. Auch wenn das nur ein Teil der Möglichkeiten der Web Services beschreibt, ist diese Aussage durchaus treffend. Web Services sind nicht für menschliche Benutzer gedacht, sondern für Softwaresysteme, die automatisiert Daten austauschen und/oder Funktionen auf entfernten Rechnern aufrufen.

Web Services orientieren sich an der Service Oriented Architecture (SOA) und vereinen daher verteilte und objektorientierte Programmierstandards und richten sich auf betriebswirtschaftliche Lösungen im Internet.

Es lassen sich die Instanzen Konsument, Anbieter und Verzeichnis identifizieren.

Der Anbieter veröffentlicht in einem Verzeichnis die Beschreibung seiner Dienste. Der Konsument durchsucht das Verzeichnis und wählt den gewünschten Dienst

aus. Nachdem eventuell weitere Protokolldetails ausgetauscht werden findet die dynamische Anbindung des Konsumenten an den Anbieter statt. Der Konsument greift nun auf Methoden zurück.

Die Grundlage hierbei bilden drei Standards, die jeweils auf XML basieren:

- UDDI als Verzeichnisdienst zur Registrierung von Web Services. Es ermöglicht das dynamische Finden des Web Services (z. B. den Dienst FussballErgebnisse) durch den Konsumenten.
- WSDL zur Beschreibung der unterstützen Methoden (z. B. Torschuetzen-Koenig) und deren Parametern (z. B. Datum) für den Programmierer.
- SOAP (oder XML-RPC) zur Kommunikation. Hier wird der eigentliche Aufruf gestartet.

Web Services bilden die drei wichtigsten Teile der Zusammenarbeit zwischen Client und Server ab: Das Zusammenfinden, Binden und den Datenaustausch.

Erreichbar sind Web Services über eine eindeutige URI. Die verwendeten plattformunabhängigen Standards sind in der Lage, entfernte Methodenaufrufe beliebiger Plattformen zu dekodieren und einer Anwendung weiterzuleiten. Auf diese Weise entsteht eine verteilte Architektur. Die Kommunikation mit Web Services erfolgt über Nachrichten, die über mehrere Protokolle transportiert werden können.

5.4.2. Vorteile

- Die verwendeten offenen Standards vermeiden jegliche Lizenzkosten. Da zu diesen Standards auch die allgegenwärtigen internetbasierten Technologien gehören, lassen sie sich auch vielerorts einsetzen. Auch hier liegt ein Kostenvorteil.
- Durch das üblicherweise verwendete HTTP-Protokoll zur Datenübertragung treten nur selten Probleme mit Firewalls auf, im Gegensatz zu vergleichbaren

Technologien wie CORBA, DCOM oder auch Java RMI. Web Services sind jedoch nicht an HTTP gebunden und lassen sich auch mit anderen Protokollen wie SMTP oder FTP übertragen und sind somit offen für verschiedene Anwendungsszenarien.

- Durch die Verwendung von bereits bestehenden und weit verbreiteten Internet-Standards (HTTP, XML etc.) entsteht eine offene und flexible Architektur, die unabhängig von den verwendeten Plattformen, Programmiersprachen und Protokollen ist. So können beispielsweise Windows-C#-Clients hinter einer Firewall mit Java-Servern, die auf Linux implementiert sind, kommunizieren. Die weit verbreiteten Standard-Protokolle ermöglichen eine Interoperabilität über jegliche Heterogenitäten im Internet hinweg.
- Die Barrieren zum Einstieg sind vergleichsweise niedrig.

5.4.3. Nachteile

- Die Hauptschwierigkeit bei der Umsetzung von Web Services dürften Sicherheitsaspekte betreffen. So ist beim Transport zu beachten, dass wichtige Web Services verschlüsselt werden oder eine Authentifizierung stattfinden kann. Ob hier HTTPS ausreichend ist oder Lösungen wie XML Signature, XML-Encryption oder SAML zu bevorzugen sind, sollte abgewägt werden.
- Ein besonderes Augenmerk liegt auf der Performanz. Diese wird durch XML, Parsen und Dateigrösse negativ beeinflusst. Der Verwaltungsaufwand nimmt bei stark verteilten Systemen zu. Der Overhead ist teilweise erheblich.
- Es ist mehr Know-How erforderlich als z.B. mit Remote Procedure Call (RPC). Programmiersprachen, mit denen man Web Services einbinden will, brauchen spezielle Bibliotheken (z. B. DOM). Schnittstellen müssen genau definiert werden - das ist viel Arbeit.
- Zudem muss der mögliche Ausfall eines für ein Unternehmen unerlässlichen

Web Services berücksichtigt werden. Eine weitere betriebswirtschaftliche Fragestellung betrifft den Bezahlmodus bei kostenpflichtigen Angeboten.

5.4.4. Einführung in Business Process Excecution Language for Web Services

Basierend auf dem Fundament der Basistechnologien SOAP, WSDL und UDDI ist es möglich, eine serviceorientierte Architektur umzusetzen, bei welcher es nicht von Interesse ist, wie die einzelnen Dienste (Services) implementiert sind.

Zur Abbildung von Prozessen auf dieser Basis wurden eine Vielzahl von Ansätzen wie bspw. die BPML (Business Process Modeling Language) Spezifikation entwickelt. Die Spezifikation mit der grössten Unterstützung von international agierenden Unternehmen stellt die BPEL4WS (Business Process Excecution Language for Web Services) dar.

BPEL4WS liefert eine Sprache für die formale Spezifikation von Geschäftsprozessen und -interaktionsprotokollen. Somit steht ein interoperables Integrationsmodell zur Verfügung, das die automatisierte Prozessintegration sowohl innerhalb von Unternehmen als auch im Bereich B2B[1]-Integration ermöglicht.

Das Initiale trio von Web Service Spezifikationen

Web Services werden als sich selbstbeschreibende Software Systeme verstanden, deren Dienste (Services) Nachrichten über internetbasierte Protokolle in einem XML-Format erhalten und senden können. Sie können verwendet werden, um eine serviceorientierte Architektur (SOA) umzusetzen.

[1] B2B: Bussiness to Bussiness

Im Folgenden wird der Fokus auf die Spezifikationen WS-Coordination, WS-Transaction und BPEL4WS gesetzt. Wie aus Abbildung 5.7 ersichtlich, können WS-Transaction und WS-Coordination unter dem Begriff Quality of Service zusammengefasst werden. Diese ermöglichen es, spezifische Standardprotokolle für Transaktionssysteme, Workflowsysteme oder andere Applikationen, die eine Koordination von Web Services wünschen, zu etablieren. Die Spezifikation BPEL4WS dient, wie in Abbildung 5.7 dargestellt, der Umsetzung von Geschäftsprozessen über Web Services. Über BPEL4WS ist eine, den Geschäftsregeln entsprechende, Komposition von Web Services möglich.

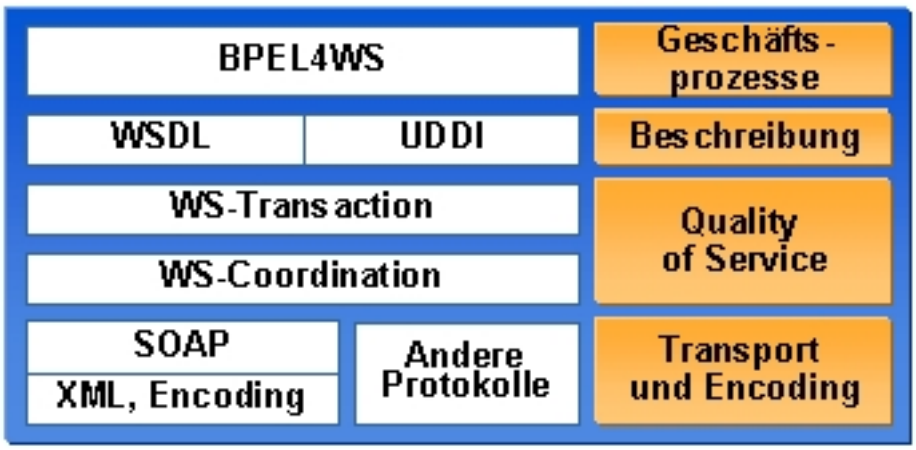

Abbildung 5.7.: Stack der BPEL4WS Spezifikationen

SOAP

SOAP beschreibt wie ein Dienst eines Web Services aufgerufen werden soll. Um eine plattformunabhängige Kommunikation zu ermöglichen, ist SOAP ein XML basiertes Nachrichtenaustauschformat. Dabei kann SOAP über verschiedene Protokolle wie beispielsweise HTTP oder SMTP übertragen werden.

WSDL

Die Web Service Description Language ist eine anbieter- und plattformunabhängige Beschreibung von Web Services. Über eine XML-Syntax können die Schnittstellen, bestehend aus den Methodennamen und den Parametern, definiert werden. Über diese Beschreibung ist es möglich, eine SOAP Nachricht für einen bestimmten Dienst eines Web Services zu erzeugen und zu verarbeiten.

UDDI

Der Hauptbestandteil der Universal Description Discovery and Integration Spezifikation ist eine zentrale Registrierung (UDDI-Registry), an welcher Web Services angemeldet werden können. Durch diese Regisitrierungsdatenbank ist es einem Web Service Client möglich, nach einem Web Service, basierend auf dessen Klassifikation und Angaben zur Firma bzw. zum Dienst, zu suchen.

WS-Coordination

WS-Coordination ist ein Framework zur Implementierung von spezifischen Koordinationsarten. Es ermöglicht, dass die beteiligten Web Services einen geteilten Kontext erhalten. Über diesen Kontext ist es möglich, dass Web Services Informationen eintragen und abfragen können.

Abbildung 5.8 stellt den in der Spezifikation vorgeschrieben Prozess zur Etablierung eines Koordinationskontextes (CC) dar.

Der Ablauf lautet wie folgt:

1. Die Applikation 1 erstellt über den Activation Service des Frameworks einen neuen Koordinationskontext (Coordinator A).
2. Die Applikation 1 sendet eine Mitteilung an die Applikation 2 im Anhang befindet sich der Coordinator A.

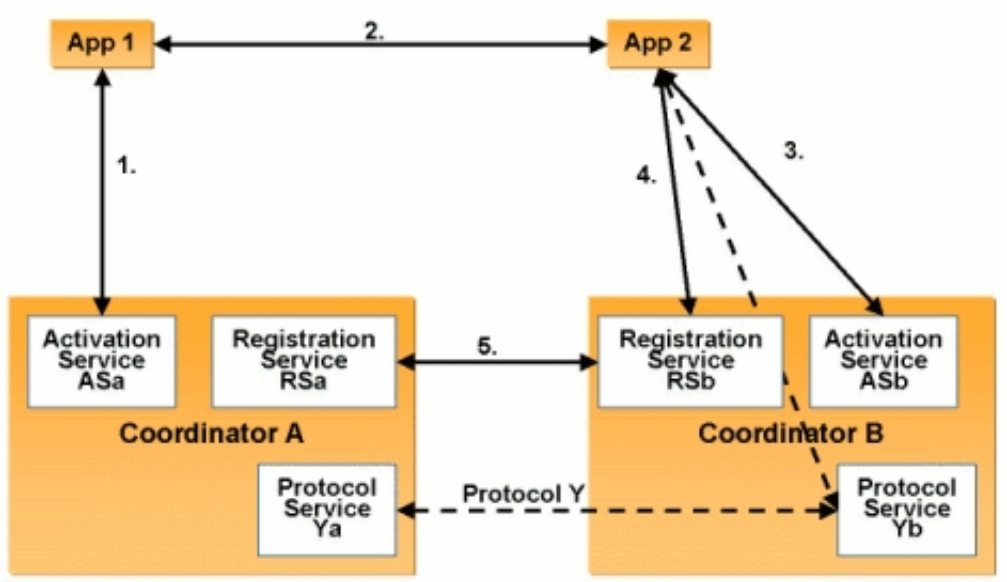

Abbildung 5.8.: Ablauf der Etablierung des Koordinationskontextes

3. Die Applikation 2 erstellt über den Activation Service einen Koordinationkontext (Coordinator B) unter der Zuhilfenahme des Coordinator A.

4. Die Applikation 2 teil ihrem Coordinator B mit, dass sie über ein spezifisches Protokoll angesprochen werden möchte.

5. Der Coordinator B teilt dem Coordinator A der Applikation 1 mit, dass die Applikation 2 über das von ihr spezifizierte Protokoll angesprochen werden muss. Das Protokoll und der geteilte Koordinationskontext gilt als etabliert.

WS-Transaction

WS-Transaction basiert auf WS-Coordination und besteht aus zwei Teilen:

- Atomic Transaction: Dieser Teil wurde von der WS-AtomicTransaction Spezifikation ersetzt
- Business Activity (WS-BusinessActivity): Definiert Protokolle für langlau-

fende Transaktionen

Die WS-Transaction Spezifikation wurde von den eigenen Kindern, der WS-Atomic Transaction und der WS-BuisnessActivity Spezifikation, überholt.

Atomic Transaction (WS-AtomicTransaction)

WS-AtomicTransaction dient der Erzeugung eines neuen Koordinationskontextes (CC) für eine neue atomare Transaktion, oder um einen zwischengeschalteten Koordinationskontext zu einer Transaktion hinzuzufügen.

Protokolle

Hierzu wird in der WS-AtomicTransaction Spezifikation eine Vielzahl von Protokollen definiert. Diese müssen von Applikationen, die diese Spezifikation implementiert haben, unterstützt werden und lauten wie folgt:

- Completion - (cp) Ein Teilnehmer (meist die Applikation, die den CC erzeugt) registriert sich für dieses, um commit oder rollback aufzurufen
- Volatile Two phase commit - (v2pc) Ein Teilnehmer, bspw. Caches, der vor der Verwendung des Durable 2PC Protokolls informiert werden möchte, registiert sich für dieses Protokoll beim CC
- Durable Two phase commit - (d2pc) Teilnehmer wie bspw. Resource Manager registieren sich für dieses Protokoll, damit der CC eine Commit-Abbruch-Entscheidung über alle Resource Manager durchführen kann

Diese Protokolle können verwendet werden, um bspw. eine Transaktion sicherzustellen. Wie in Abbildung 5.9 dargestellt:

- hat sich die Applikation 1 bei dem Koordinationskontext CoordA für das Completion Protokoll registriert,

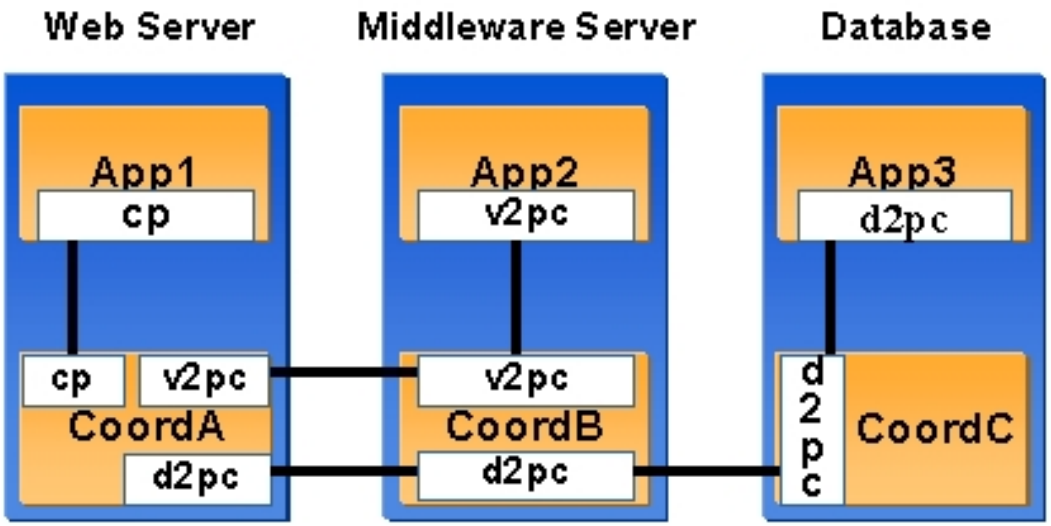

Abbildung 5.9.: Beispiel für den Protokolleinsatz bei WS-AtomicTransaction

- hat sich die Applikation 2 über ihren CoordB bei dem CoordA für das Volatile Two phase commit Protokoll angemeldet und es
- hat sich die Applikation 3 über den CoordC und CoordB bei dem CoordA für das Protokoll Durable Two phase commit registriert

Sendet die Applikation 1 an den CoordA einen Commit, informiert dieser die Applikation 2 mittels Volatile Two phase commit Protokoll über das anstehende Commit. Der Applikation 2 wird es dadurch ermöglicht ihren Cache an die Datenbank zu übertragen, bevor der Commit an die Datenbank gesendet wird. Im Anschluss daran sendet die Applikation 2 an den CoordA, über ihren CoordB, eine Mitteilung, dass die Information eingegangen und verarbeitet wurde. Nach dem Erhalt der Mitteilung, beendet der CoordA die Transaktion unter Zuhilfenahme des Durable Two phase commit Protokolls und überträgt das Commit an die Applikation 3 über den CoordB und CoordC. Im Anschluss wird der Applikation 1 mitgeteilt, dass der Commit erfolgreich durchgeführt werden konnte.

Business Activity (WS-BusinessActivity)

In einer Web Service Umgebung, kann eine Business Activity mehrere Tage dauern. Das Sperren von Ressourcen ist in diesem Fall nicht empfehlenswert. Aus diesem Grund werden Aktionen sofort durchgeführt und Änderungen an Daten sind permanent. Im Falle eines Fehlers werden Aktionen verwendet, um die schon durchgeführten Modifikationen zu kompensieren.

Eine Business Activity kann in mehrere untergeordnete Scopes aufgeteilt werden. Ein Scope ist ein Geschäftsauftrag einer beliebigen Berechnung, ausgeführt als beschränkte Menge von Operationen einer Sammlung von Web Services, die in einer bestimmten Reihenfolge ablaufen müssen. Hierzu wurden folgende Protokolle definiert.

- BusinessAgreement - Ein Teilnehmer eines untergeordneten Scopes registiert sich für dieses Protokoll mit Hilfe des vom Vorgänger übergebenen Koordinationskontextes, so dass dieser ihn steuern kann. Er bekommt unter anderem mitgeteilt, wann er seine Arbeit für eine Business Activity abgeschlossen hat und fragt, ob er die durchgeführten Tätigkeiten kompensieren muss oder sich beenden kann. Beim Fehlschlagen einer Aktivität hat er die Aufgabe, den Vorgänger zu informieren.

  Dieses Protokoll findet bei der BPEL4WS Spezifikation bei der Fehler und Kompensationsbehandlung Anwendung.

- BusinessAgreementWithComplete - Das BusinessAgreementWithComplete Protokoll ist gleichbedeutend zu dem BusinessAgreement Prokoll, mit der Ausnahme, dass ein untergeordneter Scope eine Nachricht erhält, wenn er alle Requests, die zu einer bestimmten Business Activity gehören, erhalten hat und abarbeiten kann.

BPEL4WS

Die Business Process Execution Language for Web Services (BPEL4WS) ermöglicht es, eine Menge von Web Services zu einem neuen Web Service zusammenzusetzen.

Ziel der BPEL4WS ist es, Prozesse zu beschreiben und diese falls benötigt auch zu instanziieren.

Konzepte

Zur Realisierung des Ziels werden innerhalb der BPEL4WS die unterstützenden Konzepte Variable, Fault handling, Compensation handler, Partner Link Types und Links verwendet.

- Daten für Prozessinstanzen können in der Ablauflogik referenziert werden. z.B. Nachrichen und Statusinformationen der an dem Prozess beteiligten Web Services speichern.
- Fehler können mittels eigenen Mechanismen (fault handling) abgefangen werden.
- Mittels compensation handler können jene Aktivitäten definiert werden, die duchgeführt werden sollen, wenn eine zuvor durchgeführte Aktivität nicht mehr rückgängig gemacht werden kann, aber kompensiert werden muss. Über Scopes kann der Aktionsradius dieser Kompensation definiert werden.
- Mittels Partner Link Types wird die Beziehung zwischen zwei Diensten beschrieben. Dafür wird das Konzept von Roles (Rollen) verwendet, wobei sich eine Rolle auf einen WSDL portType bezieht. Basierend auf diesen Partner Link Types können Partner Links erzeugt und innerhalb von Aktivitäten verwendet werden.

- Die Synchronsiation von einzelnen Aktivitäten kann durch das Verwenden von sogenannten Links realisiert werden. Dazu wird zunächst ein Link (¡link name=“einName”¿) definiert. Dieser Link kann nun innherhalb der Aktivität als Quelle (<source linkName=“einName”>) oder als Ziel (<target linkName=“einName”>) verwendet werden.

Umsetzung eines Geschäftsprozesses

Zur Verdeutlichung der Umsetzung eines Geschäftsprozesses, unter der Verwendung der BPEL4WS, wird hier das Beispiel der Business Process Execution Language for Web Services Version 1.1 Spezifikation gewählt.

Wie in Abbildung 5.10 dargestellt, beginnt der Prozess mit dem Eingang einer Bestellung (Receive Purchase Order). Nach dem Erhalt der Bestellung starten drei Teilprozesse. Der erste Prozess dient der Kalkulation des Preises, der zweite Prozess der Lieferungsplanung der Produkte und der dritte Prozess der Produktionsplanung der in der Bestellung angegebenen Güter. Kommen die drei Prozesse zu einem erfolgreichen Ergebnis, gilt die Fakturierung als erfolgreich bzw. der Gesamtprozess wird beendet und der Kunde wird über diesen Erfolg informiert.

Verwenden der Konzepte

Bevor der Prozess über die BPEL-Datei aufgebaut werden kann, müssen die Konzepte definiert werden, die in diesem Prozess Verwendung finden sollen.

```
<process name="purchaseOrderProcess">
      <variables>
        <variable name="PO" messageType="POMessage"/>
         ...
      </variables>
      <faultHandlers>
```

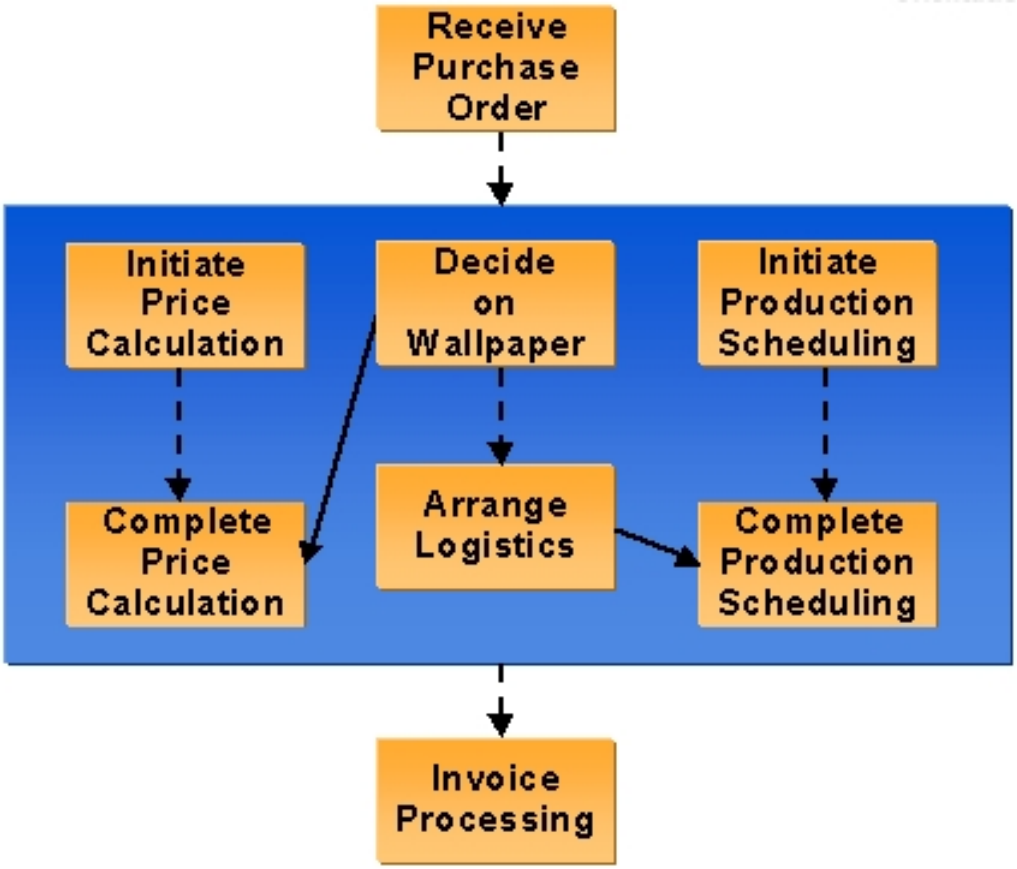

Abbildung 5.10.: Umsetzung eines Geschäftsprozesses

```
 <catch faultName="cannotCompleteOrder"
faultVariable="POFault">
   <reply partner="customer" portType="purchaseOrderPT"
     operation="sendPurchaseOrder" variable="POFault"
     faultName="cannotCompleteOrder"/>

 </catch>
</faultHandlers>
...
```

Wie in dem Listing dargestellt, sollten für den Geschäftsprozess eine Variable und ein fault handler verwendet werden. Durch die Variable ist es den Teilprozessen möglich, untereinander Daten auszutauschen bzw. Nachrichten zu senden. Über den fault handler ist es möglich, dass bei einem Auftreten eines Fehlers innerhalb eines Teilprozesses der Kunde über diesen Fehler informiert wird.

Fazit

Es wurde gezeigt, dass es über BPEL4WS möglich ist, Geschäftsprozesse unter Zuhilfenahme von Web Services zu erstellen. Hierbei stellt der resultierende Geschäftsprozess wiederum ein Web Service dar, welcher als solches in einem weiteren Geschäftsprozess Verwendung finden kann. Somit kann dieses Konzept als geschlossen gelten.

Hinzu kommt, dass die Unterstützung dieser Spezifikation weit vorangeschritten ist. Bspw. unterstützen die Softwarehersteller Oracle und IBM diese Spezifikation mit eigenen Produkten. Bei Oracle ist dies der Oracle BPEL Process Manager. Bei IBM der IBM WebSphere Business Integration Server Foundation der Version 5.1.

In den folgenden Jahren wird BPEL4WS unter dem Namen WS-BPEL wohl an Bekanntheit und Unterstützung anderer Anbieter zunehmen, da die Organization for the Advancement of Structured Information Standards (OASIS) diese Spezifikation voraussichtlich als Standard bestätigen wird.

6. Realisierung der Nutzungsdatenverwaltung

Das Datenbankschema und somit auch die Persistenzschicht wird von der Business Logik definiert. Die Business Logik wiederum muss grundsätzlich zwei Anforderungen erfüllen. Erstens muss sie die benötigten Funktionalitäten bereitstellen, damit Applikationen die angebotene Leistung in Anspruch nehmen können. Dieses sind zum einen die nötigen Mechanismen, die von den Applikationen benötigt werden, und zum anderen administrative Mechanismen, um Kunden anzulegen, zu finden, zu verändern und zu löschen. Zweitens werden Prozesse und Mechanismen benötigt, die im Hintergrund als Batch-Prozesse laufen und die Instandhaltung [1] des Systems gewährleisten. Erst nachdem diese Anforderungen definiert sind, kann das Datenbankschema entworfen und die Persistenzschicht entwickelt werden.

Es ist nötig, als erstes die Schnittstellendefinition zu entwerfen, denn diese bestimmt die nötigen Mechanismen, die von den Applikationen benötigt werden und ist somit Vorraussetzung für die Business Logik.

Die Vorgehensweise der Implementierung steht somit fest:

1. Definition der Schnittstelle und der benötigten Methoden (was wird benötigt?).

2. Definition der benötigten Mechanismen und der Business Logik (wie kann das benötigte zur Verfügung gestellt werden?).

[1] z.B. Verarbeitung von Abo´s

3. Definition des Datenbankschemas (wo werden welche Daten gespeichert?).

4. Implementierung der drei Schichten.

6.1. Schnittstellendefinition

Die Schnittstellendefinition muss, wie schon festgestellt wurde, zwei Anforderungen erfüllen. Die Bereitstellung der Mechanismen für die Applikationen und die administrativen Tasks. Die Mechanismen für die Applikationen wurden bereits durch die Darstellung der Prozessabläufe im Kapitel 5.3 weitgehend definiert. Die benötigten administrativen Mechanismen wiederum werden durch den gesunden Menschenverstand definiert. Es versteht sich von selbst, dass für eine Nutzungsdatenverwaltung z.B. Nutzungsdaten angelegt werden müssen, damit sie verwaltet werden können.

Abbildung 6.1 zeigt das Klassen-Diagramm der Schnittstellen, die bei der Realisierung der Web Services benutzt werden. Die Methoden, die aus der Sicht der Applikationen benötigt sind, werden in einer Klasse “Client” zusammengefasst. Die für administrative Aufgaben benötigten Methoden sind in der “Admin” Klasse zusammengefasst, welche die “Client” Klasse erbt.

6.2. Geschäftslogik

Die Definition der Geschäftslogik ist eine der schwierigsten und wichtigsten Aufgaben bei der Entwicklung einer Applikation. Wenn nicht genügend Zeit und Arbeit in das Design der Geschäftslogik investiert wird, führt dieses mit Sicherheit dazu, dass zu einem späteren Zeitpunkt festgestellt wird, dass eine oder mehrere Anforderungen nicht erfüllt werden können. Das System sollte nicht nur die aktuellen Anforderungen erfüllen, sondern auch offen für Zukünftige Änderungen sein. Um dies zu erreichen, wird versucht, das System so generisch wie möglich zu gestalten.

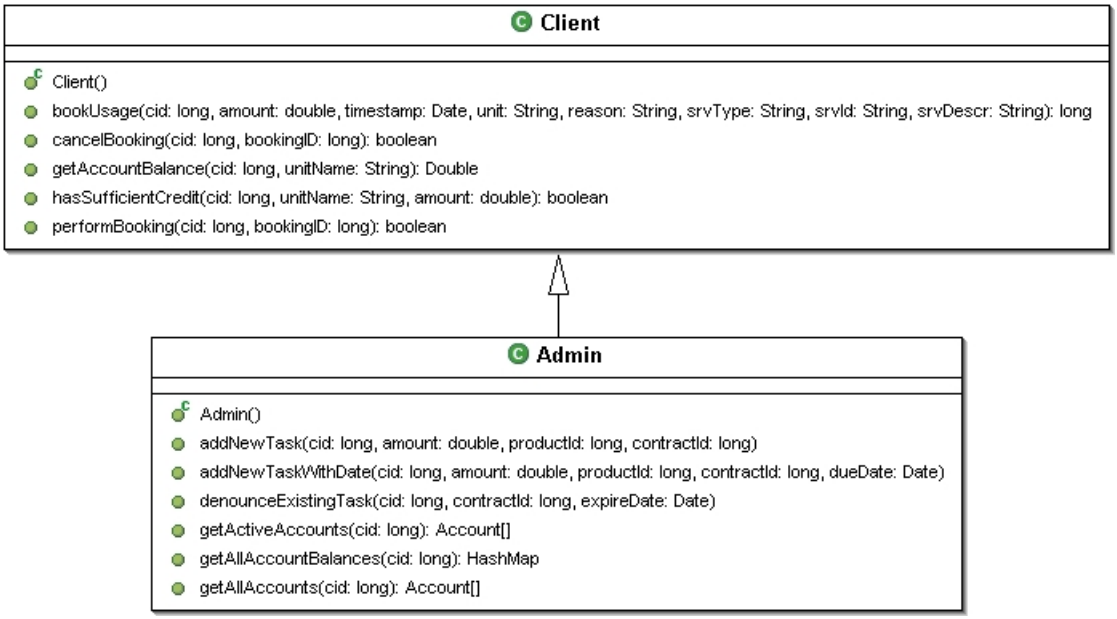

Abbildung 6.1.: Realisierung der Schnittstellendifinition

6.2.1. Kunden verwaltung

Die Kernelemente der Nutzungsdatenverwaltung sind der Kunde und die Konten der Kunden. Die Nutzungsdatenverwaltung soll von einer Vielzahl von verschiedenen Anwendungen eingesetzt werden können. Jede weitere Anwendung hat das potential eine neue Art von Verbrauchsdaten einzuführen. Die verschiedenen Arten von Verbrauchsdaten müssen separat verwaltet werden. Daraus folgt, dass jeder Kunde verschiedene Konten besitzt, von denen jede eine andere Einheit verwaltet. Einheiten werden durch den vom Kunden erworbenen Dienst spezifiziert. In dem Beispiel, dass der Kunde 50 SMS-Nachrichten erwirbt, ist die Einheit SMS.

Als weiterer Faktor kommt noch hinzu, dass von jeder Einheit verschiedene Typen existieren können. Z.B. müssen Abo-Produkte getrennt von Einmal-Produkten verwaltet werden und Bezahl-Produkte von Basic-Produkten. Der Grund ist, dass das Verhalten der Konten unterschiedlich ist. Einmal-Produkte verfallen nie, wohingegen Abo-Produkte am Ende des Abrechnungszeitraumes verfallen. Die aufgrund von Basic-Produkten erworbenen Bestände müssen beim Kauf eines Bezahl-

Produktes zurückgezogen werden.

Jeder Kunde besitzt somit eine Vielzahl von Konten. Jedes Konto ist eindeutig durch seine Einheit (SMS, EUR, ...), seine Art (Paid, Basic) und seinen Typ (once, monthly, weekly, ...) identifiziert.

Kunden

Die Kundenstammdaten werden in der Customer-Datenbank verwaltet. Die Nutzungsdatenverwaltung benötigt lediglich die Customer IDs der Kunden. Durch die zusätzliche Speicherung von Mandanten[2] und Logins werden Suchen nach Kunden, von denen die Customer ID nicht bekannt ist, vereinfacht. Ohne diese zusätzlichen redundanten Daten wäre es nötig, jedesmal zuerst mit Hilfe des Customer-Managers die Customer IDs der Logins zu ermitteln.

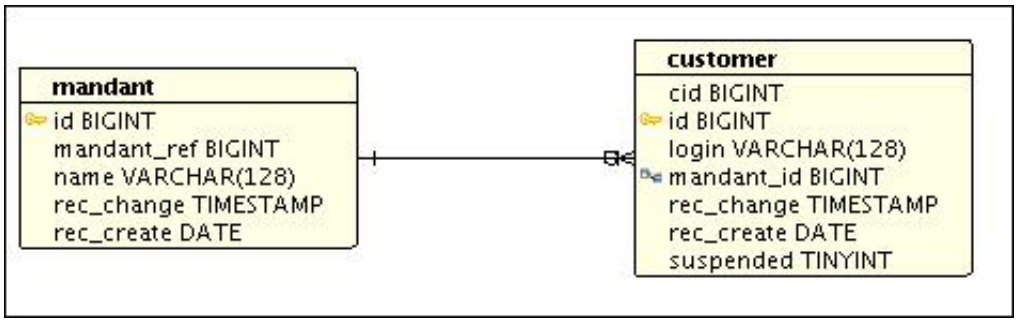

Abbildung 6.2.: Implementation der Kunden

Kundengruppen

Bezahl-Kunden sind wohl definierte Kunden, d.h. es steht zu jedem Zeitpunkt genau fest, welcher Kunde welches Produkt erworben hat. Basic-Kunden hingegen entziehen sich einer Bestimmung. Die Kundengruppe ändert sich stetig. Es

[2]Mandanten werden für die Spezifizierung von Whitelables benutzt. Die Customer IDs sind eindeutig aber mit Hilfe der Mandanten können z.B. alle freenet oder alle meome Kunden ausgewählt werden.

kommen neue Kunden hinzu oder Bestandskunden erwerben ein bestimmtes Produkt und wechseln von der Gruppe der Basic-Kunden zu den Bezahl-Kunden. Ausserdem kann jederzeit ein neues Produkt hinzukommen, welches für alle Basic-Kunden zur Verfügung steht. D.h. weder die Kundengruppe noch die Produktgruppe für diese Kunden kann genau bestimmt werden. Zudem kommt hinzu, dass der Grossteil dieser Kunden nie die Nutzungsdatenverwaltung in Anspruch nehmen wird.

Um diese Problematik zu umgehen, wird die Verwaltung der Nutzungsdaten der Basic Kunden / Produkte als On-Demand Service implementiert. Initial sind die Basic-Kunden im Gegensatz zu den Bezahl-Kunden der Nutzungsdatenverwaltung nicht bekannt. Wenn versucht wird, das Konto eines Kunden abzufragen, wird für jedes Basic-Produkt das zur Zeit angelegt ist, überprüft, ob der Kunde ein Bezahl-Kunde ist oder nicht. Wenn der Kunde kein Bezahl-Kunde ist, wird ein Konto für das Basic-Produkt angelegt. Somit wird erreicht, dass die Anzahl der Kunden und Konten minimal gehalten wird und das neue Basic-Produkte augenblicklich zur Verfügung stehen und kein zusätzlicher administrativer Aufwand entsteht. Ein Nachteil ist der durch die ständige Überprüfung entstehende overhead, der aber nicht vermieden werden kann.

Reservierung

Der erste Schritt, um eine Buchung einzugeben, ist eine Reservierung. Wenn eine Reservierung erzeugt wird, darf noch kein Verbrauch erzeugt werden, der an das Billingsystem weitergegeben wird. Ausserdem muss das Konto zwar diese Reservierung wiederspiegeln, allerdings muss die Reservierung auch wieder rückgängig gemacht werden können.

Um dies zu erreichen, werden die Reservierungen separat gehalten. Bei Anfragen des Kontostandes wird der aktuelle Stand der Konten ermittelt und die Beträge der Reservierungen, die noch nicht gebucht oder gelöscht wurden, von diesen Kontoständen subtrahiert. D.h. eine Reservierung ändert noch nichts an den Konten.

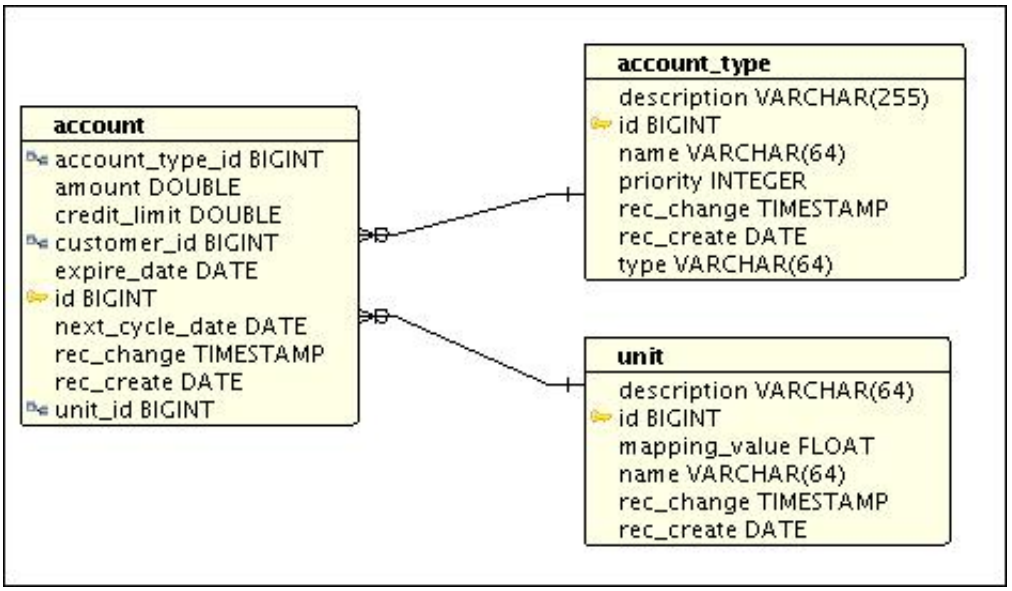

Abbildung 6.3.: Implementation der Konten

Somit kann eine Reservierung einfach rückgängig gemacht werden, indem die Reservierung als "deleted" markiert[3] wird. Erst bei einer Buchung werden aufgrund der in der Reservierung eingegebenen Daten die Konten verändert, Verbrauchsdaten erzeugt und später an das Billingsystem weitergegeben.

6.2.2. Verarbeitung von Verträgen

Wenn ein Kunde ein Produkt erwirbt und das Verbrauchskonto der Nutzungsdatenverwaltung aktualisiert werden soll, kann es zu Problemen und Ausfällen kommen, die verhindern, dass die nötigen Informationen in dem Augenblick erfasst werden können. Der Ausfall des Produkt-Kataloges ist ein solches Beispiel. In diesem Fall ist es für die Nutzungsdatenverwaltung unmöglich, die nötigen Produktinformationen zu erhalten, um zu bestimmen, was für ein Konto mit welchem Betrag aufgeladen werden soll. Ferner ist es möglich, dass die Freischaltung des

[3]Die Reservierungen werden nicht gelöscht sondern nur als gelöscht markiert damit Kundenhistorien und Auswertungen besser gemacht werden können.

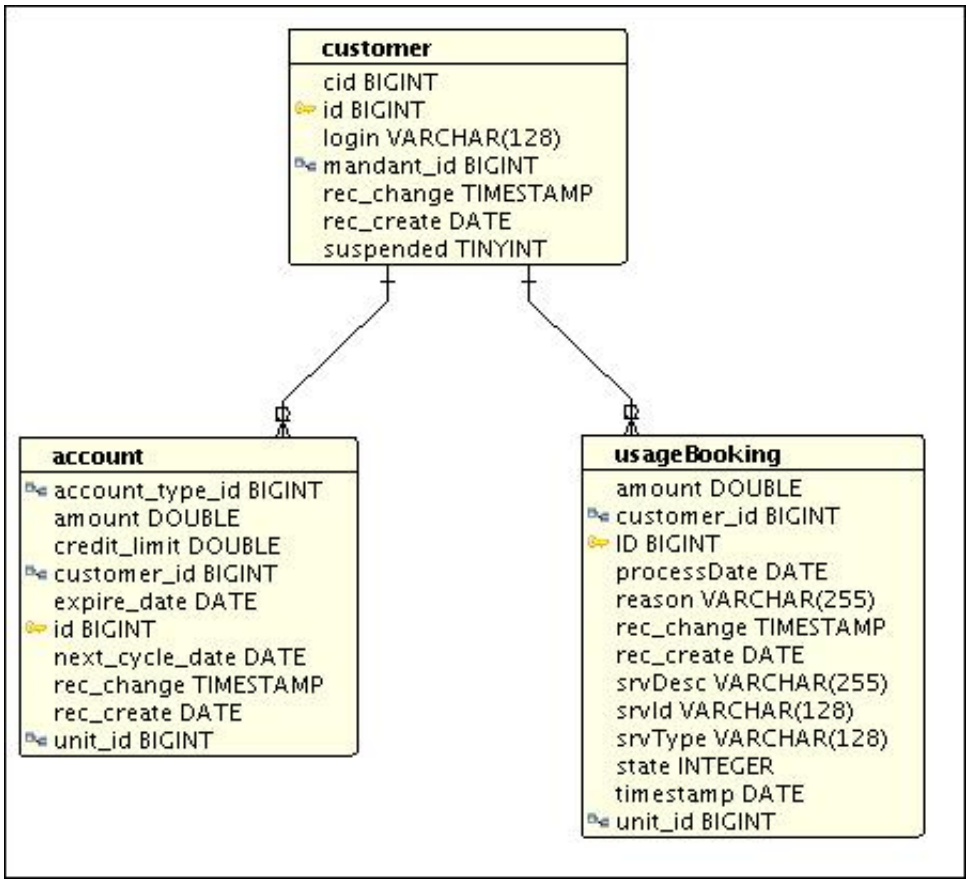

Abbildung 6.4.: Implementation der Reservierung

Produktes erst zu einem späteren Zeitpunkt erfolgen soll, weil der Kunde das Produkt erst zum ersten des nächsten Monats erworben hat. Ein weiteres Problem sind Abo Produkte. Bei diesen Produkten muss zum Anfang jedes Abrechnungszeitraumes das Konto des Kunden wieder neu aufgeladen werden.

Durch die Einführung von Tasks werden diese Probleme beseitigt. Tasks sind To-Dos die mit einem Zeitstempel versehen sind. Wenn diese Zeit erreicht wird, weiss das System, dass der Task ausgeführt werden muss. Die benötigten Informationen, um die Aufgabe zu erfüllen, sind dem Task bekannt. Wenn z.B. ein Kunde ein Produkt erwirbt, wird ein neuer Task angelegt, dem die Customer ID, die Produkt ID, der Zeitpunkt der Aktivierung usw. bekannt sind. Wenn der Task fällig wird

braucht er nur noch ausgeführt zu werden.

Die Ausführung eines Tasks geschieht zur Zeit zu zwei verschiedenen Zeitpunkten. Das erste mal wird ein Task eventuell Ausgeführt wenn er erzeugt wird. Bei der erzeugung eines Tasks wird das Fälligkeitsdatum überprüft und der Task wird Ausgeführt wenn das Datum in der Vergangenheit liegt. Unabhängig davon gibt es einen Backgroundprozess der einmal täglich Tasks, deren Fälligkeitsdatum überschritten wurde, Bearbeitet.

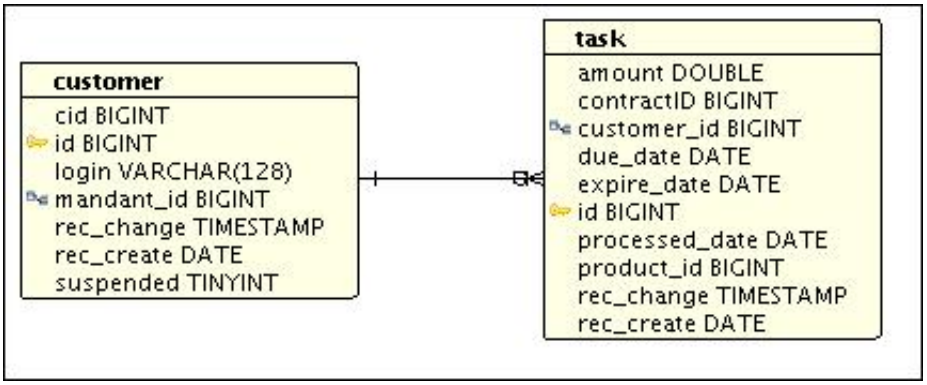

Abbildung 6.5.: Implementation der Tasks

6.2.3. Transaktionen

Alle Kontobewegungen werden als Transaktionen gespeichert. Somit wird es ermöglicht, nachzuverfolgen, was mit Kunden passiert, um eventuelle Fehler besser verstehen und beseitigen zu können. Ausserdem wird dadurch die Sicherheit des Systems erhöht und Diskrepanzen in den Kontostände besser sichtbar.

Die Verbrauchsdaten müssen mindestens einmal täglich von der Nutzungsdatenverwaltung an das Billingsystem übermittelt werden, damit das Billingsystem diese bepreisen und verrechnen kann. Ausserdem werden die Verbrauchsdaten vom Billingsystem an das Data-Ware-House, übergeben wo sie für Controllingzwecke aufgearbeitet werden. Für die Übermittlung der Daten an das Billingsystem gibt es ein spezielles, in XML definiertes Dateiformat, das eingesetzt wird. Die für die

Erzeugung der Datei nötigen Daten werden zu einem Datensatz zusammengefasst, der XDR[4] heisst und alle benötigten Informationen wie Customer ID, Service Type, Service ID usw. enthält. Aufgrund der XDRs werden CDR[5] Dateien erzeugt und an das Billingsystem übermittelt.

XDRs könnte als eine Untermenge der Transaktionsdaten bezeichnet werden. Die XDRs bestehen nur aus Verbrauchsdaten (negativ, vom Konto abgebucht), während die Transaktionsdaten jegliche Kontobewegungen beinhalten (negativ und positiv). Weil die Motivation jedoch eine andere ist sind die enthaltenen Daten unterschiedlich. Die Transaktionsdaten beziehen sich in erster Linie auf die Kontobewegungen und enthalten Daten wie aktueller Kontostand, Anderung, Grund der Änderung usw., während die XDRs die für das Billingsystem nötigen Informationen beinhalten wie Service Type und Service ID.

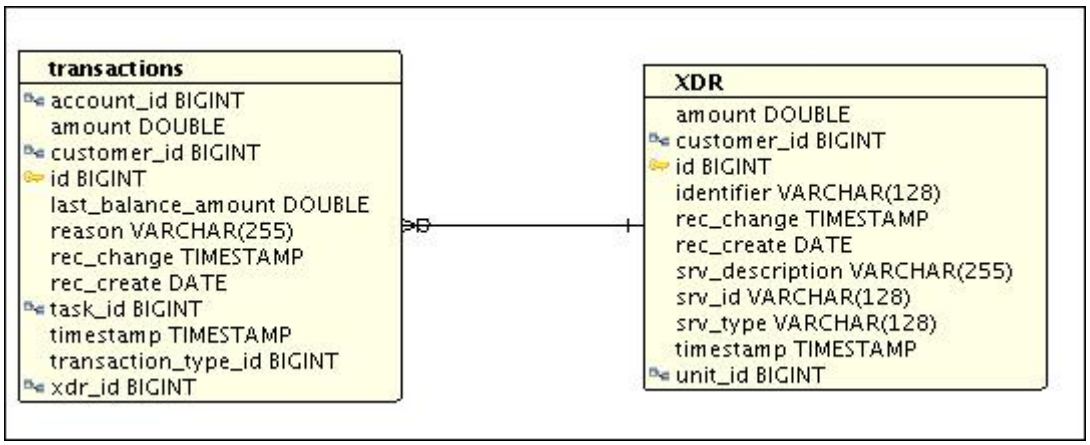

Abbildung 6.6.: Implementation der Transaktionen und XDRs

[4]XDR: eXternal Data Representation - Von Sun Microsystems entwickelter Standard zur maschinenunabhängigen Darstellung von Datenstrukturen. Er wird in RPC-Systemen benutzt und ist im RFC 1014 definiert. XDR besitzt Ähnlichkeit zu ASN.1.

[5]CDR: Common Data Representation - In IIOP (Internet-Inter-ORB Protocol) verwendete Darstellungsschicht. Der Sender von CDR-Daten verschickt diese in seinem eigenen Format und teilt dieses dem Empfänger mit. Der Empfänger ist selbst für die korrekte Umwandlung der Daten verantwortlich.

6.3. Datenbankschema

Mit Hilfe der Anforderungen und Definitionen der Business Logik ist das Datenbank Schema schnell entwickelt. Die Abbildung 6.7 zeigt das Schema aus Sicht des angebotenen Dienstes der Nutzungsdatenverwaltung, wohingegen die Abbildung 6.8 die Tabellen und Beziehungen aus Sicht des Background Prozesse für das Billingsystem darstellt.

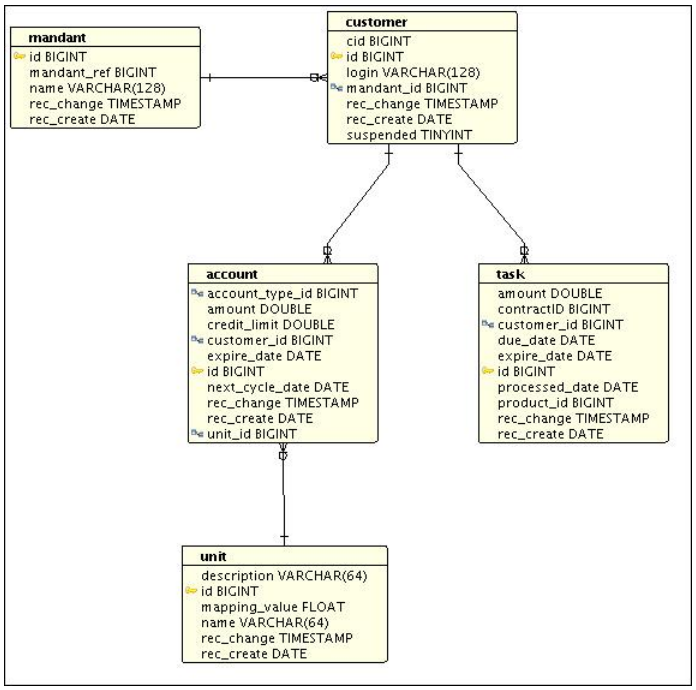

Abbildung 6.7.: DB Schema der Kunden und Konten

Am Anfang stehen die Mandanten. Jedem Mandanten sind Kunden zugeordnet, die Aufgrund der Customer ID eindeutig innerhalb der freenet.de AG Systemlandschaft identifiziert sind. Der Kunde ist mit allen anderen Tabellen verknüpft. Konten, Tasks, Transaktionen, XDRs und Reservierungen stehen in einer 1:n Beziehung zum Kunden. Somit ist der Kunde das zentrale Element von dem alle anderen Elemente der Nutzungsdatenverwaltung abhängig sind.

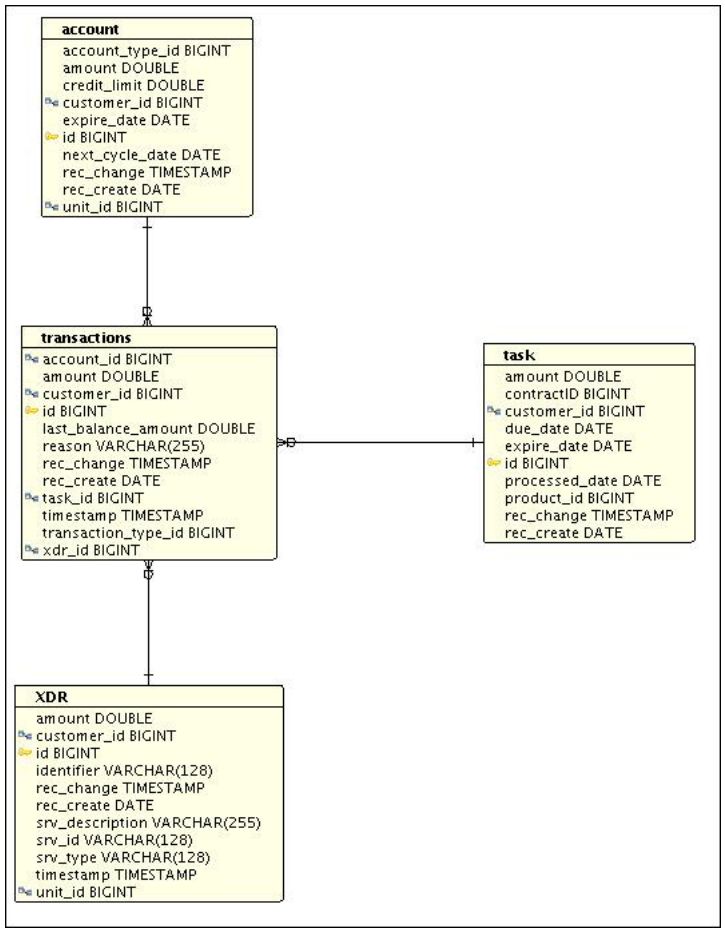

Abbildung 6.8.: DB Schema der Transaktionen und Konten

6.4. Realisierung der Anwendungslogik

Parallel zum Datenbank Schema ist das Kunden Objekt das Zentrale Element der Applikations-Logik und stellt alle Methoden zur Verfügung, die benötigt werden, um mit dem System zu arbeiten.

Kunden werden niemals explizit erzeugt. Wenn eine Anfrage für einen Kunden, der dem System noch nicht bekannt ist, gestellt wird, wird der Kunde automatisch im Hintergrund angelegt und die Konten des Kunden werden initial aufgrund der Angaben der Basic-Produkte befüllt. Vorausgesetzt ein Kunde ist dem Customer-

Manager bekannt, wird also eine Anfrage an die Nutzungsdatenverwaltung immer eine Antwort liefern.

Tasks stellen die Verträge, die zwischen dem Kunden und der freenet.de AG bestehen, dar. Folglich wird die Applikations-Logik, was ein Kunde erworben hat und was dieser Vertrag für die Nutzungsdatenverwaltung bedeutet, innerhalb der Task Objekte gehalten. Das Task Objekt holt sich aus dem Produkt Katalog die fehlenden Informationen und führt aufgrund dieser Informationen Änderungen an den Konten des Kunden aus. Wenn noch kein passendes Konto für das von den Produkteinformationen dargestellten Nutzungsdaten existiert, wird ein neues Konto angelegt.

6.4.1. Der Produkt Katalog

Die von der Nutzungsdatenverwaltung benötigten Produktinformationen (Type, Art, Einheit, Anzahl, ...) sind im Produkt Katalog noch nicht vorhanden und mussten hinzugefügt werden. Diese Informationen sind eine Art Konfigurationsparameter, die von der Nutzungsdatenverwaltung benötigt werden. Anstatt einzelne neue Felder hinzuzufügen, wurde der Produkt Katalog um "config parameter" erweitert, die eine Liste von beliebigen key / value Paaren aufnehmen kann. Durch den Einsatz von beliebigen key / value Paaren wird es ermöglicht, das auch andere Applikationen benötigte Konfigurationsparameter im Produkt-Katalog ablegen können, ohne jedesmal den Produkt-Katalog erweitern zu müssen.

6.4.2. Klassendiagramm

Das komplette Klassendiagramm ist zu gross um es in an dieser Stelle einzufügen, stattdessen wird in Abbildung 6.9 nur das Klassendiagramm der wichtigsten Komponenten dargestellt.

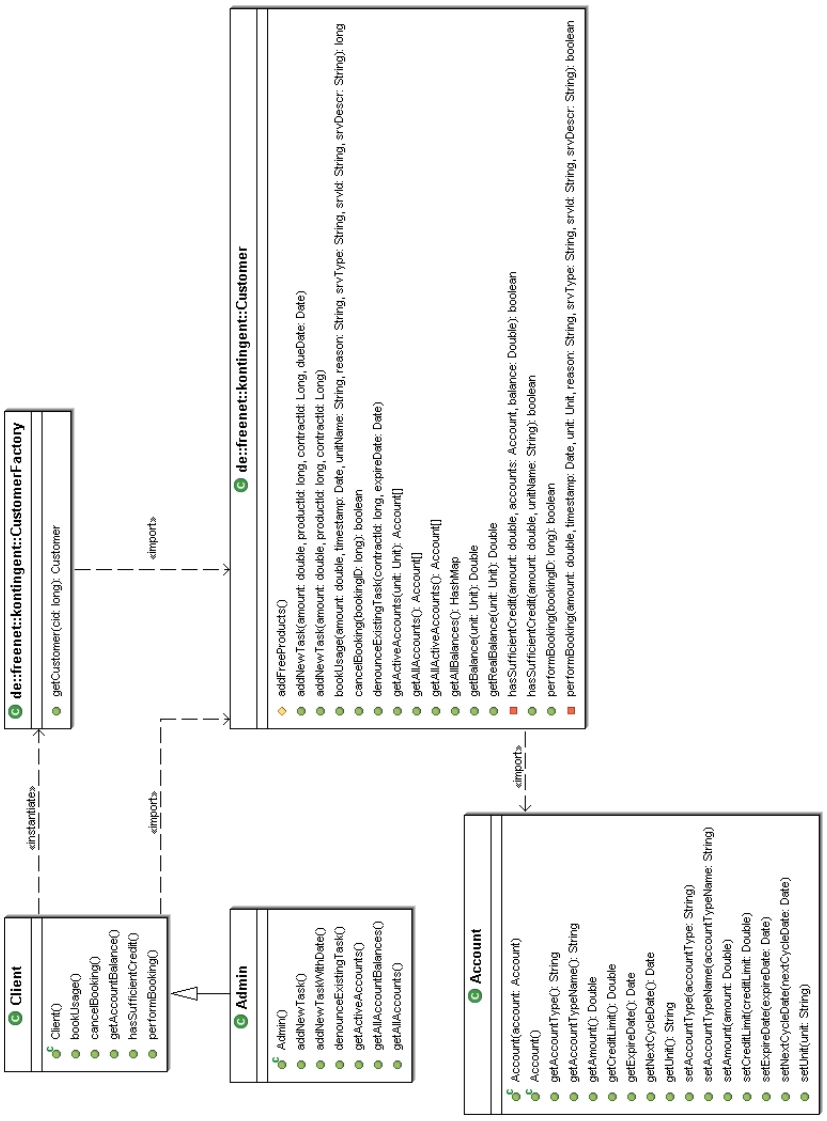

Abbildung 6.9.: Klassendiagramm der wichtigsten Komponenten

6.5. Realisierung des Web Services

Im Rahmen der Nutzungsdatenverwaltung bleibt ein gewisser Teil des Geschäftsprozesses den Applikationen überlassen. Die entwickelte Nutzungsdatenverwaltung ist ein generischer Prozess und wird von einer Vielzahl von verschiedenen Applikatio-

nen eingesetzt. Deshalb müssen z.B. die Applikationen selber entscheiden wie in Fehlerfällen verfahren werden soll.

Die eingesetzten Web Services degenerieren deshalb zu nicht viel mehr als RMI Aufrufen und ein Einsatz von den im Kapitel 5.4.4 beschriebenen Konzepten ist nicht notwendig.

Eine detaillierte Beschreibung wie Axis zu installieren ist und wie Web-Services zur Verfügung gestellt werden können, ist auf den Internet Seiten von Axis [1] zu finden. Die nötigen Schritte können wie folgt zusammengefasst werden:

1. Tomcat [16] installieren und gemäss den wünschen konfigurieren
2. Axis als webapp im Tomcat einfügen
3. Alle für den WebService nötigen Bibliotheken und Klassen in die dafür vorgesehenen Verzeichnisse unter Axis installieren.
4. WSDD Datei erzeugen
5. Im AdminClient unter Benutzung der WSDD Datei die WebServices starten

All diese Schritte, ausser die erstellte WSDD Datei, sind auf den Internet Seiten von Axis und Tomcat ausführlich beschrieben. Es sollte hier ausreichend sein, nur die WSDD Dateien abzubilden.

```
<!-- deploy.wsdd -->
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
  xmlns:ns="urn:WSNamespace">

    <service name="Client" provider="java:RPC">
        <parameter name="className"
           value="de.freenet.kontingent.soap.Client"/>
        <parameter name="allowedMethods"
```

```
            value="*"/>
    </service>

    <service name="Admin" provider="java:RPC">
        <parameter name="className"
            value="de.freenet.kontingent.soap.Admin"/>
        <parameter name="allowedMethods"
            value="*"/>
    </service>

    <!-- Wenn nicht-native Objekte (kein Date/String/Long/Integer
         usw.) in den Signaturen der WebService-Klassen enthalten
         sind m"ussen diese gemapped werden. -->
    <beanMapping qname="ns:Account"
        languageSpecificType="java:de.freenet.kontingent.Account"/>
</deployment>


<!-- undeploy.wsdd -->
<undeployment xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
  xmlns:ns="urn:WSNamespace">

    <service name="Client" provider="java:RPC">
        <parameter name="className"
            value="de.freenet.kontingent.soap.Client"/>
        <parameter name="allowedMethods"
            value="*"/>
    </service>

    <service name="Admin" provider="java:RPC">
        <parameter name="className"
            value="de.freenet.kontingent.soap.Admin"/>
```

```
        <parameter name="allowedMethods"
           value="*"/>
    </service>

    <!-- Wenn nicht-native Objekte (kein Date/String/Long/Integer
         usw.) in den Signaturen der WebService-Klassen enthalten
         sind m"ussen diese gemapped werden. -->
    <beanMapping qname="ns:Account"
        languageSpecificType="java:de.freenet.kontingent.Account"/>
</undeployment>
```

Mit Hilfe des AdminClients und entsprechenden WSDD Dateien ist es also möglich, präzise zu bestimmen, welche Methoden von welchen Klassen unter einem WebService bereitgestellt werden sollen. Im Rahmen der Nutzungsdatenverwaltung wurden Wrapper-Klassen erzeugt, die als Ansatzpunkt für die WebServices gelten sollten. Aus diesem Grund werden einfach alle Methoden innerhalb der WSDD eingebunden, weil die Trennung, welche Methoden zur Verfügung gestellt werden sollen, bereits in den Klassen gemacht wurde.

7. Zusammenfassung und Ausblick

Die Aufgabenstellen der Entwicklung und Implementation eines zentralen Dienstes zur Integration von Nutzungsdaten heterogener Dienstplattformen konnte mit Ausnahme von zwei potentiellen Problemquellen erfolgreich erfüllt werden.

Die erste Fehlerquelle befindet sich in der Datenhaltung und Verarbeitung der Konten von Abo-Produkten. Wenn ein Kunde zwei oder mehrere gleiche oder sehr ähnliche (gleiche Einheit, gleicher Abrechnungszyclus) Abo-Produkte erwirbt, werden die erworbenen Kontingente auf das gleiche Konto aufgeladen. Die Verarbeitungsweise eines Abo-Produktes ist so, dass am Ende des Abrechnungszeitraumes das Konto des Kunden auf einen bestimmten Wert aufgeladen wird, unabhängig davon wie der aktuelle Kontostand lautet. D.h. Kontingente, die von dem Kunden eventuell nicht in Anspruch genommen wurden, verfallen beim Ablauf des Abrechnungszeitraumes. Einmal Produkte hingegen sind kumulierend und werden auf den aktuellen Kontostand aufaddiert.

In dem Fall, dass der Kunde mehrere ähnliche Abo-Produkte besitzt, wäre das richtige Verhalten, dass dieser Zustand vom System erkannt und das Konto dementsprechend aufgeladen wird. Ein kleines Beispiel soll dieses verdeutlichen. Der Kunde erwirbt erstens ein Monatsabo für 25 SMS-Nachrichten und zweitens ein Monatsabo von 50 SMS-Nachrichten. Der richtige Kontostand am ersten des Monats wäre 75 SMS-Nachrichten. In unserem Fall wäre der Kontostand allerdings entweder 25 oder 50 SMS-Nachrichten, je nachdem, welches Produkt als letztes verarbeitet wird, weil das letzte Produkt den aktuellen Kontostand überschreibt.

Es ist möglich, das System so zu erweitern, dass die Erkennung und Verarbeitung von mehreren Abo-Produkten ermöglicht wird. Weil die aktuelle Marketing-

und Verkaufsstrategie der freenet.de AG zum Zeitpunkt der Implementierung dieser Diplomarbeit das erwähnte Fehlerszenario nicht erlaubt und in absehbare Zeit auch nicht erlauben wird, wurde zugunsten der Abrechnungsgeschwindigkeit und Systemressourcen entschieden, diese potentielle Fehlerquelle fürs erste zu vernachlässigen. Das System ist allerdings so ausgelegt, dass eine Erweiterung zur Lösung dieses Problemfalles ohne grössere Probleme möglich sein sollte.

Die zweite Fehlerquelle entsteht aufgrund einer endlichen Rechengeschwindigkeit und Systemressourcen und betrifft wieder die Abo-Kunden. Theoretisch muss das Konto eines Abo-Kunden am Ende seines Abrechnungszeitraumes augenblicklich wieder aufgeladen werden, so dass Verbrauchsdaten im neuen Abrechnungszeitraum auch von dem Bestand des neuen Abrechnungszeitraumes abgebucht werden. Praktisch ist es so, dass mit steigender Anzahl von Abo-Kunden die Verarbeitung aller Verträge mehr und mehr Zeit in Anspruch nehmen wird, so dass es in diesen Zeiträumen zu Problemen kommen kann.

Um dieses Problem einzugrenzen, wurden zwei Massnahmen implementiert. Die erste Massnahme ist, dass Konten ein Gültigkeitsdatum besitzen. D.h., nachdem der Abrechnungszeitraum beendet ist, sind Automatisch alle Konten inaktiv bis die Verarbeitung der Tasks ausgeführt und die Konten wieder neu aufgeladen wurden. Der Vorteil dieses Vorgehens ist, dass zumindest keine falschen Kontostände entstehen können, aber der Nachteil ist, dass in der Zwischenzeit der Kunde den Service nicht in Anspruch nehmen kann, obwohl er dazu berechtigt wäre. Die zweite Massnahme ist, dass die Konten der Bezahl-Kunden als erstes verarbeitet werden. Weil die Anzahl der Bezahl-Kunden deutlich geringer ist als die Anzahl von Basic-Kunden, wird durch die Sortierung erreicht, dass der Zeitraum in dem die Bezahl-Kunden den Service nicht benutzen können, minimiert. Die Bereitstellung des Services an Basic-Kunden ist eine Werbeleistung. D.h., selbst wenn es einige Verzögerungen gibt, können sich die Basic-Kunden im Grunde nicht beschweren, denn schliesslich ist es Kostenlos.

Literaturverzeichnis

[1] Axis: Apache implementation des SOAP standarts der W3C.
`http://ws.apache.org/axis/`

[2] Cayenne: Professionelles Objekt Relational Mapping Framework.
`http://www.objectstyle.org/cayenne/`

[3] Michael Kifer, Arthur Bernstein, Philip M. Lewis. *Database Systems: An Application-Oriented Approach*, 2nd Edition.
Addison Wesley, 2005.

[4] Eclipse: an open extensible IDE for anything and nothing in particular.
`http://www.eclipse.org/`

[5] webMethods Glue 6.0. Enterprise Services Platform.
`http://www.webmethods.com/`

[6] Hibernate: Hibernate ist ein Framework, das zur Persistenz von Java-Objekten eingesetzt wird.
`http://www.hibernate.org/`

[7] Ian F. Darwin. *Java Cookbook*. Solutions and Examples for Java Developers.
O'REILLY, 2001.

[8] Leap NextGen: Billingsystem der Formula Telecom Solutions.
`http://www.fts-soft.com/leap_adjunct.html`

[9] Bernd Oestereich. *Objektorientierte Softwareentwicklung.*
Oldenbourg Verlag München Wien, 2001.

[10] Martin Fowler. *Refactoring.*
Addison-Wesley Professional, 1999.

[11] Programmer to Programmer *Professional Apache Tomcat 5.*
Wiley Publishing, Inc, 2004.

[12] W3C. W3C SOAP Specifikation.
`http://www.w3.org/TR/soap/`

[13] Jakarta Tapestry: Ein open-source Framework für die erstellung von dynamischen, robusten und skalierbaren Web-Applikation.
`http://jakarta.apache.org/tapestry/`

[14] Stefan Edlich, Patrick Kunert. *Tapestry: Webanwendungen mit dem Apache Framework.*
Software & Support Verlag, 2004.

[15] The Server Side. Your Enterprise Java Community.
`http://www.theserverside.com/`

[16] Tomcat: Apache Tomcat implementation eines servlet kontainers für Java Servlets und JavaServer Pages technologie.
`http://jakarta.apache.org/tomcat/`

[17] Tom Pender. *UML Bible.*
John Wiley & Sons, 2003.

[18] Gustavo Alsonso, Fabio Casati. *Web Services: Concepts, Architectures and Applications.*
Springer Verlag, 2004.

[19] World Wide Web Consortium (W3C). Extensible markup language (xml) 1.0 (third edition).
`http://www.w3c.org/TR/REC-xml/`

[20] Mark Birbeck *et al. Professional XML, 2nd Edition.*
Apress, 2004.

[21] Chris Boar. *XML Web Services in the Organizasion.*
Microsoft Press, 2003.

A. Wichtige Komponenten der freenet.de AG Systemlandschaft

A.1. Point Of Sale

Der Point Of Sale ist die zentrale Applikation für die Aufnahme von Bestellungen. Die verschiedenen Shops sind nur für die Auswahl der Produkte zuständig. Die Aufnahme der Kundeninformationen und Bezahldaten sowie zusätzlichen Produktabhängigen Daten wird am Point Of Sale vorgenommen.

A.1.1. Überblick

Der Point Of Sale besteht aus einer Produktiv und einer Stand-By Datenbank sowie sieben Applikationsserver, die jeweils einen Apache WebServer und eine Tomcat instance besitzen. Die POS Applikationen sind Tapestry [13] Web-Applikationen.

Die Applikations-Logik ist in Java Klassen ausgelagert, so dass in den einzelnen Anwendungen lediglich das Design und Sonderbehandlungen implementiert werden müssen.

Eine Standart Bestell-Applikation hat 5 Arbeitsschritte, die durchlaufen werden. Am Ende ist die Bestellung des Kunden abgeschlossen und wird zur weiteren Verarbeitung an das Order-Management-System weitergegeben.

Die Arbeitsschritte sind:

1. HomePage: diese Seite wird dem Kunden nicht angezeigt. Es werden die vom Shop übergebenen Parameter gelesen und ein Bestellprozess wird initiert.
2. Bestellübersicht und LoginPage: auf dieser Seite muss sich der Kunde authentifizieren.
3. Kundendaten: Eingabe der Adressdaten sowie Bankdaten.
4. Sonderbehandlung: in Abhängigkeit des Produktes werden an dieser Stellt eventuell zusätzliche Eingaben getätigt.
5. Bestellbestätigung: die Bestellung wird dem Kunden nochmal angezeigt und bestätigt.

A.1.2. Die Struktur

Der Eingangspunkt zu dem System ist ein Switch. Der Switch verteilt die Anfragen auf einen der vorhandenen WebServer. Die WebServer entscheiden, um welche Applikation es sich handelt und an welchen Tomcat diese Anfrage weitergeleitet werden soll. Somit wird ein im hohen Masse skalierbares System erzeugt, das den Anforderungen entsprechend angepasst werden kann.

A.2. Customer Manager

Die freenet.de-Kundendatenbank ist der zentrale Bestandteil der Kundenstammdatenverwaltung. Sie enthält alle kundenrelevanten Stammdaten wie Adress-, Rechnungs-, Zahlungsinformationen, usw.

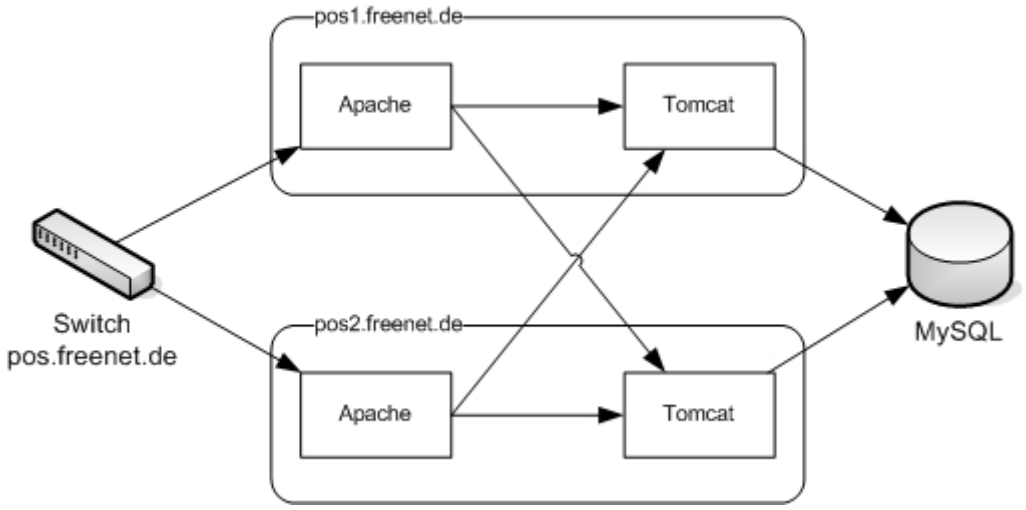

Abbildung A.1.: Struktur des Point Of Sales

A.2.1. Überblick

Grob unterteilt besteht die Kundendatenbank aus den produktiven Datenbanken, sowie einer Replikationsdatenbank pro produktiver Instanz zur Hot-Stand-By-Absicherung.

Der Zugriff auf die Datenbank ist durch eine Middle-Tier gekapselt. Derzeit kommen hier die Protokolle SOAP und RMI zum Einsatz. Direkte Zugriffe werden somit im Produktivbetrieb vermieden. Eine Besonderheit bildet bei diesem verteilten System die Vergabe der Kundennummern, der sogenannten CID. Wird ein neuer Kunde in der Datenbank angelegt, so wird mittels einer eigens implementierten C/S-Applikation - CID-Server - eine eindeutige Nummer erzeugt. Die Server-Applikation verteilt hierbei Nummernbereiche an n-Clients. Die Slaves können diese Nummernpakete einzeln an die jeweilig anfragende Applikation vergeben und sind somit unabhängig von der Datenbank und einem möglichen Ausfall des Servers.

A.2.2. Die Struktur

Derzeit sind jeweils neun Maschinen für Applikationsserver, Produktiv- und Replikationsdatenbank im Einsatz (siehe Abbildung A.2). Das Gesamtsystem ist auf eine weitreichende Skalierung ausgelegt. Die SOAP/RMI-Requests an die Applikationsserver werden über den Switch geloadbalanced. Die Kunden werden mittels eines Verteilungsschlüssels auf die einzelnen Kundendantebanken verteilt. Jeder Kunde hat eine eindeutige, rein numerische Kundennummer - die CID. Für diese wird ein module 256-Wert ermittelt. Das Resultat wird mittels einem Mapping auf eine der Produktivdatenbanken gemappt. Jede dieser Produktivdatenbanken wird genau einmal mittels eines Replikationsmechanismus auf genau eine weitere Datenbank repliziert. Dies ist lediglich zur weiteren Absicherung, um eine mögliche Ausfallzeit so kurz wie möglich zu halten. Das Replique ist für den paralellen Produktionsbetrieb nicht bestimmt und wird nur beim Ausfall des Produktivsystems aktiviert. Für die Umschaltung eines Slaves zu einem Master sind ein paar Handgriffe notwendig.

A.2.3. Die Applikationsserver

Die Applikationsserver sind die einzige Schnittstelle um auf die zentrale Kundendatenbank mittels einer Applikation zuzugreifen. Hierfür stehen die Protokolle SOAP und RMI zur Verfügung. Die dahinterstehenden Applikationen sind reine Java Entwicklungen.

A.3. Order-Management- oder auch Workflow-System

Das Freenet Workflow-System wurde im Rahmen des Acoreus-Blacklist Systems erstellt und mit dem Blacklist System erfolgreich implementiert.

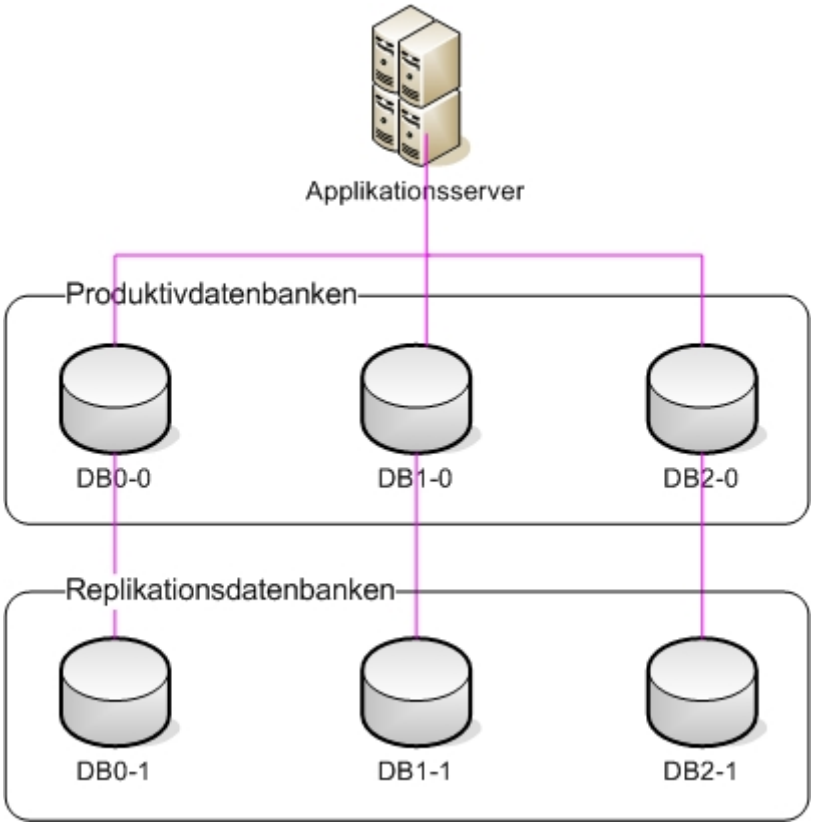

Abbildung A.2.: Grobe Struktur des Customer Managers

Da Freenet ein Order-Management-System (oder auch Workflow-System) für die Prozesssteuerung (z.B. Auftragsabwicklung) und System-Integrationsaufgaben benötigt, wurde dieses System für diese Anforderungen ausgebaut.

Grundzüge des Workflow-Systems

- Java basiert (1.4.1).
- MySQL Datenbank für Speicherung von Vorgängen (Tasks) und Workflow-Definitionen (BusinessCases).
- Als SOAP Engine wird Apache Axis zusammen mit dem Web Application

Server Tomcat verwendet.

- Ein BusinessCase besteht aus einer Ansammlung von sog. Step-Handlern, welche eine Aktivität für einen Schritt im Workflow definiert. BusinessCases können dabei in mehreren Versionen vorliegen.
- Das Routing basiert auf der Definition von sog. StepTransitions, welche ein Resultat eines StepHandlers auf den darauf folgenden Step mappen.
- Eine Aktivität eines StepHandlers ist eine Java-Klasse, welche ein einfaches Interface implementiert. Diese Klasse führt die Aktivität durch oder initiiert eine entsprechende Aktivität (z.B. durch einen SOAP-Aufruf). Das Ergebnis ist ein numerischer Return-Code, der für das Routing verwendet wird sowie optional Veränderungen and den Task-Daten (Task-Parameters).
- Zusätzlich gibt es die Möglichkeit, manuelle Steps als Teil des Workflows zu definieren, welche von Mitarbeitern mit definierten Rollen manuell bearbeitet werden können.
- Tasks werden in der Regel durch SOAP-Aufrufe in das Workflow-System eingestellt.
- Die Verarbeitung erfolgt durch einen oder mehrere WorkflowQueueProcessoren.
- Durch eine (derzeit Kommandozeilen-) Applikation kann Einfluss auf die Tasks genommen werden. Diese Applikation benutzt intern SOAP-Calls, die auch von anderen Applikationen genutzt werden können. Derzeit sind Teilfunktionen im CIS-Teil des Blacklist-Systems abgebildet.
- Zur Definition von Workflows steht eine Kommandozeilen-Applikation zur Verfügung, die mithilfe einer einfach-strukturierten Textdatei die Definition manipulieren lässt. Unterstützt wird dies durch die graphische Darstellung von BusinessCases mit Hilfe des frei verfügbaren AT&T Graphwiz-Packages (DOT).

- Steuerung und Monitoring des Workflow-Systems durch JMX (Java-Management eXtentions).

A.3.1. Struktureller Überblick

Abbildung A.3 zeigt einen Überblick des Order-Management-Systems.

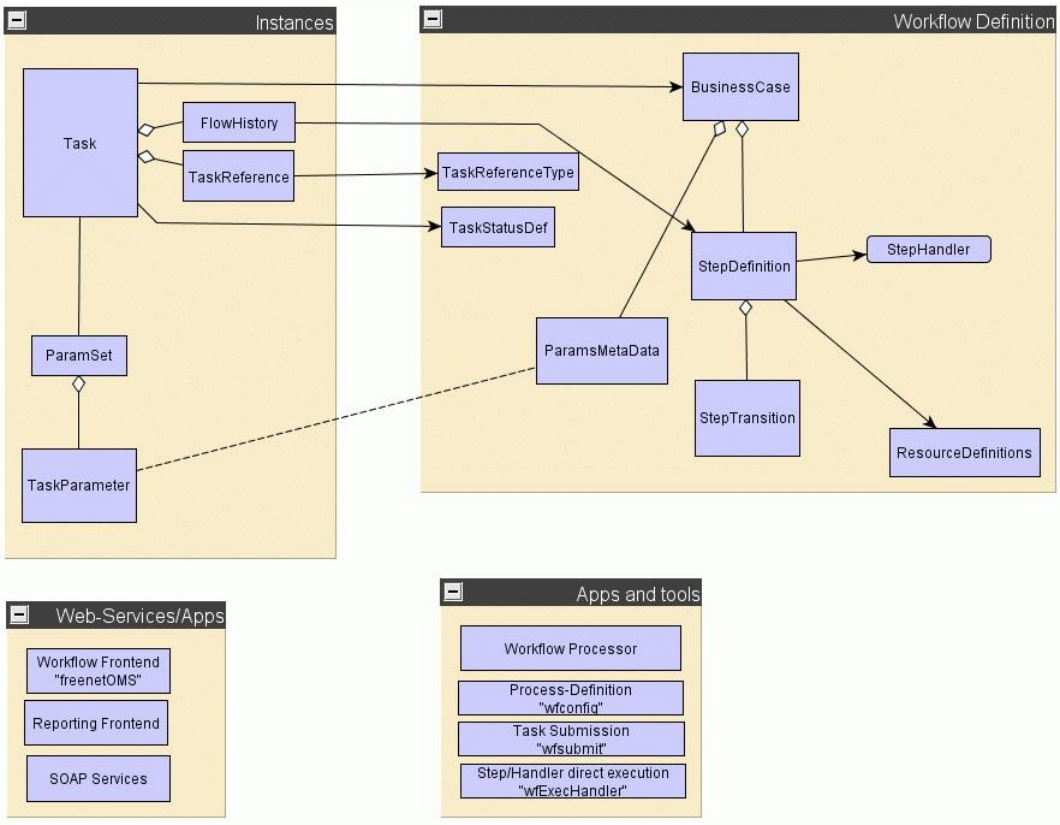

Abbildung A.3.: Struktureller Überblick des OMS

A.4. Produkt Katalog (PCat)

Der Produkt Kataloge stellt eine zentrale Verwaltung aller Produkte und Tarife in einer dezentralen Systemlandschaft zur Verfügung. Der Sinn und Zweck ist es, dass Applikationen und Dienstleistungen nur noch mit Produkt IDs zu arbeiten brauchen und jegliche weiteren Informationen wie den Preise, Produkttexte, Service Types usw. aus dem Produkt Katalog auslesen können. Ausserdem wird somit sichergestellt, dass keine Inkonsistenzen zwischen den verschiedenen Systemen entstehen und Produktinformationen nur noch an einer Stelle geändert werden müssen.

A.4.1. Aufgaben und Funktionen

- Produktnamen & -nummern
  - für verschiedene Sichten
- Gültigkeiten & Versionen
  - für verschiedene Sichten
- Eigenschaften (Texte, Parameter, ...)
  - für verschiedene Sichten
- Abhängigkeiten
  - z.B. “Produkt A immer nur im Zusammenhang mit Produkt B aber nie zusammen mit Produkt C”

“Für verschiedene Sichten” bedeutet, dass die Produktinformationen, die vom Produkt Katalog geliefert werden in Abhängigkeit davon, wer diese Anfrage gestartet hat, unterschiedliche Inhalte beherbergen kann.

Wenn ein Produkt z.B. mit verschiedenen White-Labeln verkauft werden soll, würde der Produktname je nachdem, welches System die Information anfordert, den für das White-Label zugehörigen Namen enthalten. Das gleiche gilt natürlich auch für die anderen Eigenschaften, Abhängigkeiten usw.

A.5. Billingsystem (LNG)

Die freenet.de AG setzt als Billingsystem das von dem Unternehmen Formula Telecom Solutions entwickelte System Leap NextGen [8] ein. LNG ist eine Komplettlösung für Online-Verrechnung, Kundenverwaltung und Rechnungsstellung. Es ermöglicht es, seinen Benutzern kreative Geschäftsprozesse und Marketingideen zu modellieren. Durch eine einzigartige Angehensweise unterstützt LNG sowohl Prepaid als auch Postpaid innerhalb einer zusammenlaufenden Umgebung.

Entgegen herkömmlichen Billingsystemen, welche hart codierte oder Tabellen basierte Regeln benutzen, bietet LNG eine einzigartige Engine für das erzeugen von Services und Business Regeln. Somit wird es ermöglicht, die Business Logik des Systems komplett innerhalb der GUI Oberfläche zu designen und zu konfigurieren.

Die Featureliste von Leap NextGen ist zwar nicht unendlich, kommt dem doch sehr nahe. Im Folgenden sind nur die Kernfeatures dargestellt. Für weitere Features und Eigenschaften wird auf die Webseiten von LNG [8] verwiesen:

- Service Konvergenz: Unterstützung jeglicher Service Typen mit flexibler Definition der Attribute pro Service Type. Der Businessplaner stellt Services zusammen und erstellt spezifische Kundenprodukte.
- Pre-Post Konvergenz: Postpaid, Prepaid sowie hybride Pre-Postpaid Kunden werden innerhalb eines einzigen Systems unterstützt.
- Beliebige Business Regeln: Die Rating und Business Regel Engine basiert auf einem patentiertem Decision Operation Tree (DO Tree) Konzept, eine

sehr flexible Engine die Fortgeschrittene Features und Business Regeln unterstützt.

- Standart Integration: Leap NextGen ist ein offenes System das eine komplette API zu allen seinen Modulen bietet und Standart Protokolle unterstützt.
- Leistung und Architektur: Leap NextGen setzt eine n-tier Architektur sowie verteilte CORBA middle-ware ein und unterstützt 100% web basierte Clients.
- Einkommenssicherung: Ein komplettes Prüfungswesen, Controlling und Berichterstattung ermöglicht das Verfolgen aller Systemprozesse. Durch die hohe Leistungsfähigkeit und das Echtzeitvermögen wird sichergestellt, dass Nutzungsdaten und Events schnell verarbeitet werden. Somit wird die Zeit zur Rechnungsstellung reduziert sowie gutes Nutzungsmonitoring und Fraud Kontrolle ermöglicht.
- Administration und Sicherheit: Einer Administration und Systemsicherheit wird ein grosser Wert bemessen. Tools zum Monitoring, Prüfungswesen, Sicherheitsprofiele Fehlerkorrektur und Berichtswesen sind vorhanden.

A.5.1. Der Decision Operation (DO) Tree

Das Rating, welches von LNG eingesetzt wird, basiert auf dem einzigartigen und patentierten Modell der DO Trees. Ein DO Tree besteht aus Nodes und Branches. Jeder Node beinhaltet einen Entscheidungsmechanismus, welcher bestimmt, welcher Branch verfolgt werden soll und Aktionen, welche bestimmen, welche Aktivitäten ausgeführt werden sollen, wenn ein Node erreicht wird.

Ein DO Tree hat folgende Vorteile:

- Definition von Businessregeln in einem natürlichen und logischem Arbeitsfluss

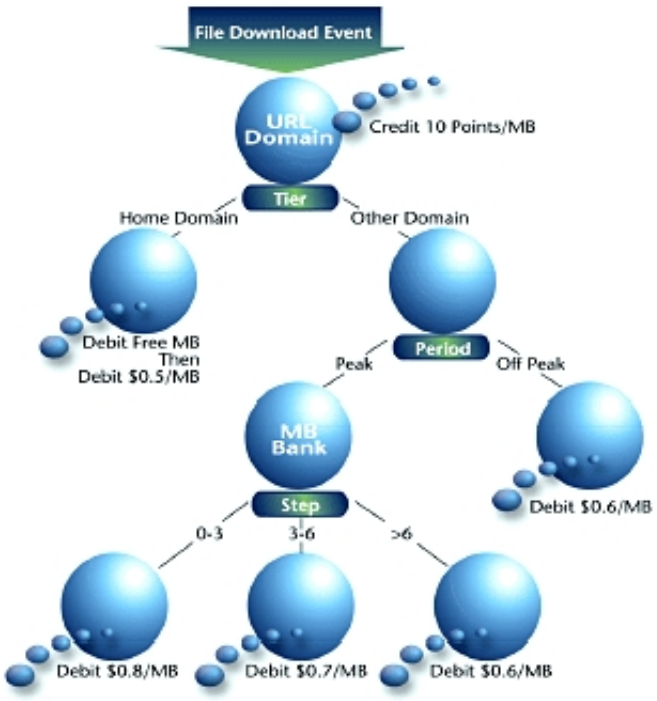

Abbildung A.4.: Beispiel eines DO Trees

- Sogar die komplexesten Businessregeln können in einem DO Tree abgebildet werden
- Eine einzelne Regelstruktur kann alle Fasetten eines Events behandeln (z.B. Provisionierung, Benachrichtigung über Erfolg und Berechnung der Aktion)
- Unabhängige Attribute. Der DO Tree kann beliebige Attribute, sei es ein berechneter Wert, Kunden Daten, Event Attribute usw., erfassen und bei dem Entscheidungsprozess verwenden.
- Unanhängige Aktionen. Der DO Tree kann beliebig viele Aktionen eines beliebigen Types auslösen.

B. Technik und Tools

B.1. Axis

Apache AXIS ist eine open source Implementierung des Web Service Standards SOAP unter der Lizenz der Apache Software Stiftung. AXIS kann für die Entwicklung von Clients, Servern und Gateways verwendet werden. Das Hosting von Web Services, d.h. der Betrieb und das Anbieten von Web Services kann ebenfalls mit AXIS durchgeführt werden. AXIS bietet die folgenden Highlights:

- Open Source unter der Lizenz der Apache Software Stiftung
- Hohe Performanz
- Flexible Konfiguration und Erweiterbarkeit
- Unabhängigkeit vom Transport. Es kann HTTP, FTP oder Mail verwendet werden.
- Unterstützung von SOAP 1.1 und teilweise von 1.2
- WSDL Unterstützung über Code Generatoren
- JAX-RPC und SAAJ Konformität
- Erweiterungen für Sicherheit
- Provider für den Zugriff auf Enterprise JavaBeans
- ...

B.1.1. Entstehung

AXIS ist ein Nachfolgeprojekt der beliebten Apache SOAP Implementierung, die aus dem IBM Projekt SOAP4J hervorging. Die Architektur von AXIS wurde von Grund auf neu gestaltet. Apache SOAP basiert noch auf dem langsamen Dokument Objekt Modell DOM. AXIS verwendet dagegen SAX Filterketten. Bezogen auf Funktionalität, Performanz und Interoperabilität hat AXIS ab der Version 1.0 die Apache SOAP Implementierung bereits überrundet. Ein wesentlicher Vorteil von AXIS ist die konsequente Ausrichtung an die JAX-RPC Spezifikation von Sun.

Für AXIS wurde nicht der Name Apache SOAP Version 3.0 verwendet, da zum damaligen Zeitpunkt eine Unsicherheit über die Bezeichnung der SOAP Technologie beim W3C bestand. Der Name AXIS wurde mit der Bedeutung Apache eXtensible Interaction System möglichst nichtssagend gehalten.

B.1.2. Architektur

AXIS ist keine starre Software aus einem Guss. Module gliedern AXIS in Subsysteme, die verschiedene Aufgaben übernehmen. Die Verarbeitung von Anfragen erfolgt über Filter ketten, die sich aus Handlern zusammensetzen.

Die Verkettung der Handler ist flexibel und kann über Konfigurationsdateien angepaßt werden. Über eigene Handler kann in die Verarbeitung von SOAP Nachrichten eingegriffen werden. Typische Aufgaben für die Handler sind Logging oder Sicherheitsüberprüfungen.

Bei der Anwendungsentwicklung ist es nicht notwendig selbst in die Filterketten einzugreifen. Mit dem Konzept der Ketten ist AXIS jedoch offen für Erweiterungen.

Damit Web Services bereit gestellt werden können, wird ein Server benötigt, der Web Anfragen verarbeiten kann. AXIS kann in einen Web Container wie den Tomcat integriert werden. Es gibt dafür eine Web Anwendung, die AXIS in Form von

drei Servlets beinhaltet. Abbildung B.1 zeigt einen Web Container, in welchem die drei Servlets installiert wurden.

Das AXIS Servlet kann Web Services aufnehmen und Anfragen an diese zur Weiterverarbeitung routen. Aufgrund der Zustellungsfunktion spricht man von einem SOAP Router oder Dispatcher.

Das Admin Servlet dient der Fernwartung des SOAP Routers. Web Services können über dieses Servlet installiert und deinstalliert werden.

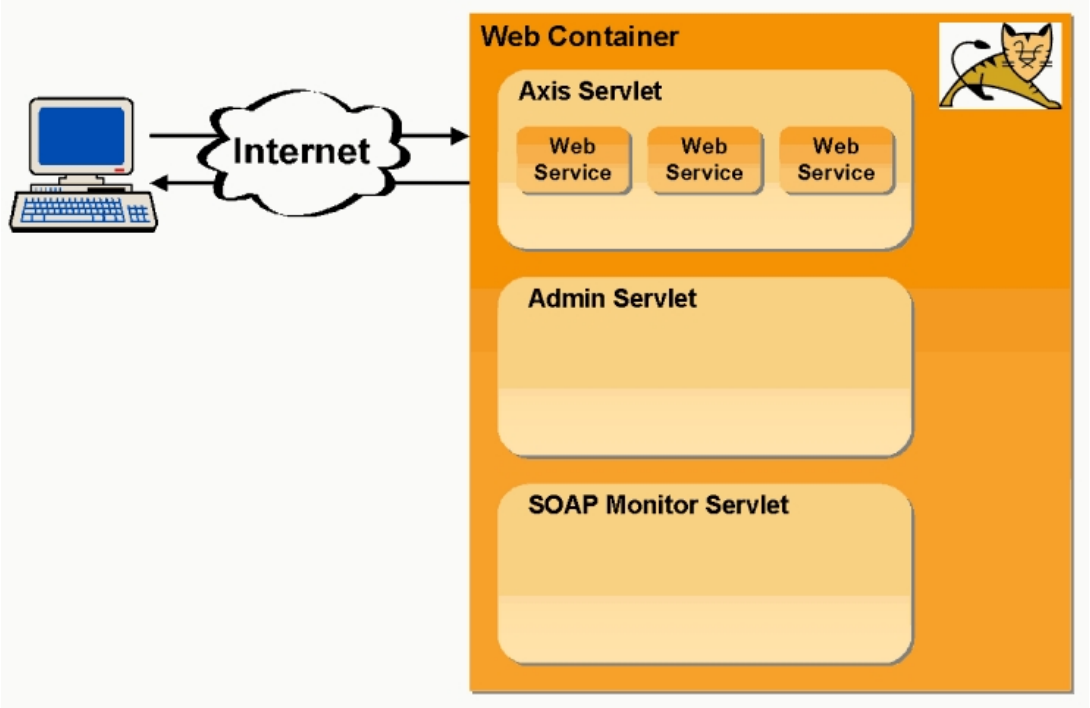

Abbildung B.1.: AXIS Engine im Web Container

B.1.3. Was ist Deployment?

Um einen Web Service zu nutzen muß dieser zuvor im SOAP Router installiert werden. Man bezeichnet diesen Vorgang als Deployment. Ein installierter Web

Service hat zur Laufzeit bestimmte Eigenschaften, die während des Deployments festgelegt werden können. In einem Deploymentdescriptor können diese Eigenschaften festgelegt werden. Für den Deploymentdescriptor wird eine XML Datei verwendet.

B.1.4. SAAJ und JAX-RPC Konformität

Die Verwendung von SOAP basierten Web Services mit Java wird mit Spezifikationen und Schnittstellen beschrieben. JAX-RPC definiert den Aufruf von entfernten Methoden während SAAJ den Zugriff auf SOAP mit Attachments regelt.

Nach dem Bestehen der Compliance Tests für JAX-RPC und SAAJ von Sun wurde die Version 1.0 von AXIS freigegeben.

Was ist SAAJ?

SAAJ oder SOAP with Attachments API for Java ist eine Spezifikation von Sun. AXIS implementiert die Schnittstellen dieser Spezifikation und verwendet SAAJ als Basis.

Was ist JAX-RPC?

JAX-RPC beschreibt Schnittstellen und Konventionen für entfernte Funktionsaufrufe. Das Mapping von Java Datentypen auf XML Schema Typen, die von SOAP verwendet werden, wird beschrieben. Für die Generierung von Java Klassen aus WSDL Dateien und die Erzeugung von WSDL aus Java Klassen gibt es Vorschriften.

B.1.5. Interoperabilität

Wichtig für die Verbreitung von Web Services ist die Interoperabilität zwischen den verschiedenen Tools und Plattformen. Ein interoperabler Web Service kann ohne spezielle Anpassungen von anderen Plattformen benutzt werden. Beispielsweise kann ein mit Java realisierter Web Service von .NET Clients verwendet werden.

Zu welchen C++, Perl, Python, etc. SOAP Implementierungen AXIS kompatibel ist, kann dem SOAP Builder Vergleich entnommen werden. Dort sind Testergebnisse aufgeführt, die bis zu den einzelnen Datentypen die Interoperabilität zwischen den Tools auflisten.

B.1.6. WSDL Unterstützung

AXIS kümmert sich um die Codierung und Decodierung von XML Nachrichten. Der Entwickler kann sich ganz auf seine Anwendung konzentrieren.

WSDL oder die Web Services Description Language beschreibt einen Web Service so, dass mit den Informationen dieser Beschreibung eine komfortable Verwendung eines Web Service möglich wird. Neben der Adresse des Servers enthält WSDL auch eine Beschreibung der Funktionen und der zugehörigen Daten für einen Aufruf.

WSDL ist nicht auf eine Programmiersprache oder eine Plattform begrenzt. Eine WSDL Beschreibung eines mit C# realisierten .NET Web Service kann für die Erzeugung eines passenden Clients mit Java verwendet werden.

Automatische WSDL Generierung

WSDL Dateien können für installierte Web Services abgefragt werden. Die WSDL Beschreibung des Web Services wird automatisiert erzeugt. Zum Abfragen reicht ein einfacher Web Browser.

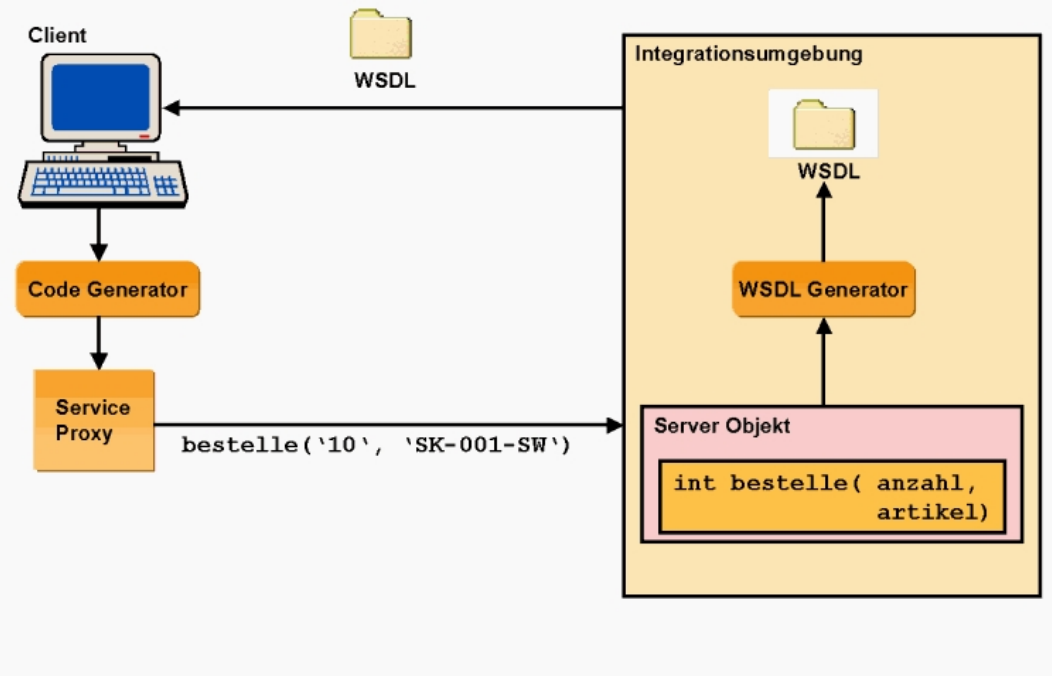

Abbildung B.2.: Generierung von Proxys aus WSDL

Erzeugen von Client Klassen aus WSDL

Aus einer WSDL Beschreibung eines Web Services können mit dem WSDL2Java Tool Klassen für Client und Server erzeugt werden. Für den Server werden Rumpfklassen erzeugt, mit deren Hilfe sich leicht Serverfunktionalität ansprechen läßt. Mit den Client Klassen kann mit wenigen Zeilen Code ein Client für einen bestehenden Web Service erstellt werden.

Erzeugen von WSDL aus Java Code

WSDL Dateien sind recht kompliziert aufgebaut und eignen sich nicht für eine Erstellung von Hand. Das AXIS Werkzeug Java2WSDL kann aus normalen Java Klassen eine WSDL Datei generieren. Die WSDL Datei kann über einen Web Server veröffentlicht werden, um Fremden die Nutzung eines Web Services zu er-

leichtern.

Ausgehend von einer WSDL Datei kann mit einer leistungsfähigen Entwicklungsumgebung in Minuten ein Client für einen unbekannten Web Service erzeugt werden.

B.1.7. Anforderungen

Für eine AXIS Installation wird Java ab der Version 1.3 und ein Servlet 2.2 konformer Web Container benötigt. Die Entwickler von AXIS empfehlen die Verwendung von Tomcat.

B.1.8. Sicherheit

AXIS bietet die Möglichkeit die Client Anfrage und die Antwort vom Server mit SSL zu verschlüsseln. Darüber hinaus kann AXIS mit dem XML-Security Projekt von Apache eingesetzt werden, um Teile einer SOAP Nachricht zu verschlüsseln. So kann der Body einer Nachricht, der die Nutzdaten enthält verschlüsselt werden, während der Header unverschlüsselt bleibt, um ein Routing zu ermöglichen. Mit AXIS wird ein Codebeispiel ausgeliefert, welches den Einsatz von XML-Security für die Signierung des SOAP Umschlages demonstriert.

B.1.9. Zugriff auf Enterprise JavaBeans

Methoden eines stateless SessionBeans können als Web Services angeboten werden, ohne eine Zeile Code zu schreiben. Es genügt ein Deployment Descriptor, der den Web Service beschreibt. Im Descriptor wird der Name des Beans und seine Schnittstellen angegeben. Mit dem Deployment Descriptor kann ein Web Service installiert werden, der Zugriff auf das Enterprise JavaBean ermöglicht.

Abbildung B.3 zeigt wie EJBs als Web Service angeboten werden können. Der Application Server mit den Enterprise JavaBeans kann sich sogar auf einem anderen Rechner befinden, als der Web Service. Schickt der Client eine Anfrage an den Web Service, wird diese entgegengenommen und an den Application Server delegiert. Für den Application Server ist es ein Aufruf wie jeder andere auch. Er verarbeitet die Anfrage und liefert einen Rückgabewert, der an den Web Service geschickt wird. Der Web Service verpackt den Wert in eine SOAP Nachricht im XML Format und sendet diese zurück an seinen Client.

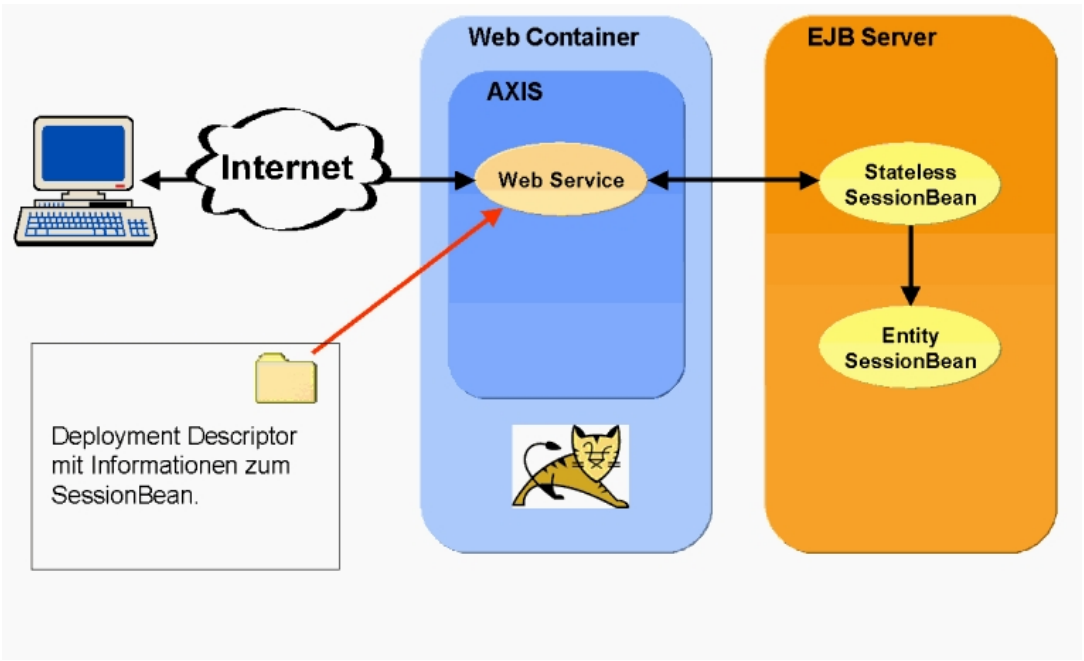

Abbildung B.3.: Enterprise JavaBeans als Web Service

B.1.10. Tools

Neben den WSDL Generatoren sind im AXIS Paket weitere Werkzeuge für den Entwickler enthalten.

Monitoring

Der TCP Monitor kann zwischen Client und Server geschaltet werden. SOAP Nachrichten, die zwischen Server und Client ausgetauscht werden, können über eine graphische Oberfläche verfolgt werden. Die ausgetauschten Nachrichten sind dank XML sowohl für Maschinen als auch Menschen lesbar. Der Monitor ist für Debugging und für die Suche nach Interoperabilitätsproblemen hilfreich.

B.1.11. Fazit

Die W3C Standards SOAP 1.1 und in Teilen die kommende SOAP Version 1.2 sowie WSDL werden von AXIS wie die Java Standards JAX-RPC und SAAJ unterstützt.

Werkzeuge wie die WSDL Generatoren und schnelles Deployment über das Admin Interface verhelfen dem Entwickler zu mehr Produktivität bei der Entwicklung von Web Services. Detailierte Kenntnisse der SOAP Spezifikation sind für das Verwenden und die Erstellung von Web Services nicht mehr notwendig.

AXIS ist ein würdiger Nachfolger von Apache SOAP, der mit vielen Features, einer ausgereiften Architektur und mit Standardkonformität glänzen kann. AXIS bietet mit seiner ausgereiften Architektur und der Open Source Lizenz eine gute Basis für Anwendung und Web Service Tools.

B.2. Cayenne

Dieses Kapitel präsentiert Cayenne - eine schnelle, skalierbare und einfach zu lernendes open source Objekt/Relational Mapping (ORM) Framework. Cayenne ist eine seltene art von open source Projekt. Es ist nicht nur eine “editiere dein eigenes XML” runtime Bibliothek sondern ein Integriertes Packet, das ein Modellierungs- und Entwicklungstool beinhaltet. Dieses Kapitel zeigt, wie eine ORM Applikation

mit Cayenne einfach Generiert wird und beschreibt die Kernfeatures und Design Prinzipien.

B.2.1. Was ist Cayenne

Cayenne wurde im Jahre 2001 als ein open source Projekt gestartet. Es ist ein starkes Objekt/Relational mapping Framework der Qualität bezahlpflichtiger Produkte. Eine partielle liste von Features von Cayenne beinhaltet:

- Inkrementelles abrufen von Daten und “Lazy Relationships”.
- Objekt-Abfragen, inklusive Sortierung und Filterung von Objekten im Speicher mit der Cayenne Abfrage-Sprache.
- Isolation von Objekte-Änderungen zwischen Benutzer Sessions.
- Ausführung von allen neu erzeugten Objekten, Änderungen oder gelöschten Objekten mit einem einzigen commit.
- Verteiltes caching.
- Automatisches ordnen von DML Operationen zur Sicherstellung von inkonsistenzen der Datenbank.
- Verschmelzung von mehreren Datenbanken in eine virtuelle Datenquelle
- Sicherstellung der Unabhängigkeit von Datendanken durch Adapter für alle gängigen Datenbanken.
- Verschiedene Strategien zur automatischen Generierung von primary key´s
- ... und viele mehr

Das ”Warenzeichen” von Cayenne ist seine einfache Handhabung, das schnelle Erlernen und die Entwickler Produktivität. Die Idee, die die Entwicklung von

Cayenne stark beeinflusst hat, war, dass kein Benutzer ein weiteres XML Format erlernen möchte, geschweige denn manuell mapping Dateien editieren möchte. Während es möglich ist, die mapping-Dateien von Cayenne manuell zu editieren, ist es nur sehr selten notwendig. Der CayenneModeller wird standartmässig mit Cayenne zusammen vertrieben und stellt ein GUI tool zur Verfügung, das jegliche mapping und Entwicklungsoperationen unterstützt. Der CayenneModeller vereinfacht das Erlernen von Cayenne für Anfänger und steigert die Produktivität von fortgeschrittenen Anwendern. Es hilft der Visualisierung von Datenbank-Schemen und Objektschichten und unterstützt sowohl die Wiederverwendung parametrisierter Abfragen als auch schnelles Prototyping.

Cayenne versteht die Notwendigkeit, dass es eine Anzahl von wiederkehrenden Operationen gibt, welche auch ausserhalb der GUI implementiert werden müssen. Z.B. vergessen viele Benutzer die Neuerzeugung der persistenten Java Klassen, nachdem Änderungen am mapping im CayenneModeler vorgenommen wurden. Ein anderes Beispiel ist, dass Applikationen in verschiedenen Umgebungen (z.B. Entwicklungs-, Testing-, Produktivumgebung) eingesetzt werden und es unproduktiv wäre, alle Konfigurationen jedesmal, wenn ein neues Release entwickelt wird, im GUI anpassen zu müssen. Für diese Fälle bietet Cayenne Ant Tasks zur Integration in build Dateien an.

B.2.2. Cayenne Einführung

Erzeugung eines Projektes

Zur Demonstration verschiedener Cayenne Features wollen wir uns eine einfache Kunst Katalog Datenbank ansehen. Abbildung B.4 stellt das Schema der Datenbank für den Katalog dar.

Typische Cayenne Projekte bestehen aus einem mapping und Datenbank-Zugriffobjekten (beide werden durch XML Dateien beschrieben die mit dem CayenneModeler erzeugt werden). Ein Projekt besteht aus DataMaps und DataNodes, die in

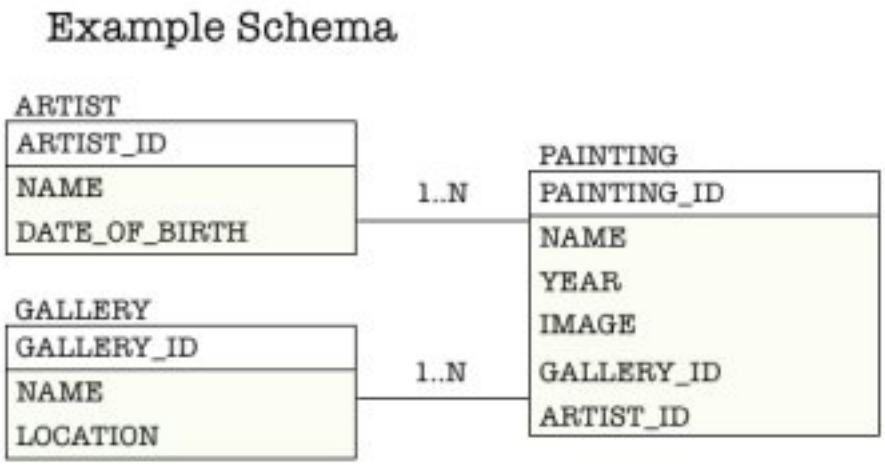

Abbildung B.4.: Schema der Kunst Katalog Datenbank

einem DataDomain organisiert sind. DataMaps ist ein Repository aller mapping Informationen der Datenbank und des Objekt Layers. DataNode ist eine Abstraktion einer einzelnen physikalischen Datenbank. Verschiedene Datenbanken können in einem Projekt vermischt werden, z.B. MySQL und Oracle DataNodes können in einem Projekt koexistieren. DataNodes brauchen nicht JDBC konform oder relational sein, allerdings sind sie im Normalfall beides. Es ist möglich DataNodes für andere Datenarten zu erzeugen, wie z.B. Dateien oder LDAP. DataDomains fügen ein oder mehr DataNodes und DataMaps zu einer virtuellen Objekt-Relationalen Datenquelle zusammen. Intern werden die Abfragen zu den zuständigen Datenquellen verteilt. Somit kann Cayenne ein mapping für einem sehr Komplexes physikalisches Layout von Datenbank Servern herstellen

Mit nur wenigen Schritten kann eine persistente Objektschicht aus einer existierenden Datenbank generiert werden:

1. Starten sie CayennModeler und erzeugen sie ein neues Projekt.
2. Klicken sie auf “DataDomain”, um eine neue DataDomain anzulegen.
3. Selektieren sie diese DataDomain und klicken sie auf “DataNode”, um einen

DataNode Kindobjekt anzulegen.

4. In der Konfigurationseingabe für den DataNode selektieren sie "DriverDataSourceFactory" als "DataSource Factory", wählen sie den passenden Datenbankadapter und geben sie die nötigen Verbindungsinformationen an.

5. Selektieren sie aus dem Menu "Tools / Reengineer Database Schema" und folgen sie den Anweisungen des Wizards. Dieses wird ein DataMap erzeugen, welches alle Datenbank Tabellen enthält und die dazugehörigen Java Klassen.

6. Wählen sie diesen DataMap aus und selektieren sie aus dem Menu "Tools / Generate Classes" um die persistenten Java Klassen anzulegen.

Dieses vorgehen nimmt ca. 2 Minuten in Anspruch und erzeugt ein voll funktionales set an persistenten Java Klassen, die zum Einsatz in Applikationen einsetzbar sind. Cayenne persistente Klassen sind POJO (also "Plain Old Java Objects") und benutzen keine bytecode "Verbesserungen". Die Standart-Klassen-Generation-Templates (welche auf Velocity basieren und einfach anzupassen sind) generiert die folgenden Kind-/Super- Klassenpaare für die ARTIST Tabelle:

```
/** Class _Artist was generated by Cayenne.
  * It is probably a good idea to avoid changing this class
  * manually, since it may be overwritten next time code is
  * regenerated. If you need to make any customizations,
  * please use subclass.
  */
public class _Artist extends
             org.objectstyle.cayenne.CayenneDataObject {

    public static final String ARTIST_NAME_PROPERTY
           = "artistName";
    public static final String DATE_OF_BIRTH_PROPERTY
           = "dateOfBirth";
    public static final String PAINTING_ARRAY_PROPERTY
```

```
            = "paintingArray";

    public static final String ARTIST_ID_PK_COLUMN = "ARTIST_ID";

    public void setArtistName(String artistName) {
        writeProperty("artistName", artistName);
    }
    public String getArtistName() {
        return (String)readProperty("artistName");
    }

    public void setDateOfBirth(java.util.Date dateOfBirth) {
        writeProperty("dateOfBirth", dateOfBirth);
    }
    public java.util.Date getDateOfBirth() {
        return (java.util.Date)readProperty("dateOfBirth");
    }

    public void addToPaintingArray
                (org.objectstyle.art.Painting obj) {
        addToManyTarget("paintingArray", obj, true);
    }
    public void removeFromPaintingArray
                (org.objectstyle.art.Painting o) {
        removeToManyTarget("paintingArray", obj, true);
    }
    public List getPaintingArray() {
        return (List)readProperty("paintingArray");
    }
}
```

sowie

```
public class Artist extends _Artist {

}
```

Die Superklasse enthält die “set” und “get” Methoden aller Properties und sollte Grundsätzlich nie manuell editiert werden. Die Kindklasse enthält keinen sourcecode und wird bei nachfolgenden Ausführungen der Klassen-Generierung nicht wieder geändert / überschrieben. Durch diesen Ansatz ist sichergestellt dass der generierte Code nie mit dem selber entwickeltem Code in Konflikt tritt. Somit sind die Kindklassen der ideale Platz für eigenständige Logik im Zusammenhang mit dem Applikations-Domain-Modell als auch für Business Logik.

Applikations-Code

Die Persistenz aller Operationen wird durch Instanzen vom DataContext gewährleistet. Der Benutzer muss sich beinahe nie mit einer potentiell komplexen Struktur von DataDomains und DataNodes auseinandersetzten. Stattdessen bietet der DataContext eine Fassade zum Rest der Cayenne Zugriff-Schicht. Änderungen an Objekten, die noch nicht ausgeführt wurden, werden von Änderungen durch andere DataContexte isoliert. In einem multi-user Applikationsumfeld wird der DataContext gewöhnlich innerhalb des Session Scopes gehalten und führt somit dazu, dass laufende Änderungen innerhalb der Session privat bleiben.

Hier nun ein paar Beispiele herkömmlicher Methoden, einen DataContext zu erzeugen.

- Command-line und Swing Applikationen können eine einfache Statische Methode Benutzen:

  ```
  import org.objectstyle.cayenne.access.DataContext;
  ...
  DataContext context = DataContext.createContext();
  ```

- Web Applikationen registrieren normalerweise einen speziellen listener in der web.xml, so dass ein DataContext automatisch in den Session Scope hinzugefügt wird:

```
<!-- Inside web.xml -->
<listener>
   <listener-class>
      org.objectstyle.cayenne.conf.WebApplicationListener
   </listener-class>
</listener>


// WebApp Java Code
import org.objectstyle.cayenne.conf.BasicServletConfiguration;
import org.objectstyle.cayenne.access.DataContext;
import javax.servlet.http.HttpServletRequest;
...
HttpServletRequest r;

// retrieve DataContext stored in the session
// by the WebApplicationListener
DataContext c = BasicServletConfiguration.
                getDefaultContext(r.getSession());
```

Die am häufigst benutzten Arbeitsschritte persistenter Objekte sind:

- Abfrage der Datenbank um eine Liste von Objekten zu erhalten, die bestimmten Kriterien entspricht.
- Herbeiholen von verwandten Objekten.
- Änderung von Objekteigenschaften.
- Löschen von persistenten Objekten.

- Anlegen von neuen persistenten Objekten.
- Ausführen von commits, um Änderungen an die Datenbank weiterzuleiten.

Bei diese Arbeitsschritten Hilft Cayenne ungemein, indem der nötige Code reduziert und die Produktivität der Entwickler gesteigert wird. Die folgenden zwei Beispiele zeigen wie wenig code benötigt wird um all die genannten Arbeitsschritte zu Benutzen.

```
// Fetch all artists who worked in the XX century..
// note how easy it is to build property expressions
Expression qualifier =
   Expression.fromString
              ("paintingArray.year between 1899 and 2000");
SelectQuery query = new SelectQuery(Artist.class, qualifier);
List artists = context.performQuery(query);

if(artists.size() > 0) {
  Artist firstArtist = (Artist) artists.get(0);

  // "Simple" properties are accessible via generated
  // "get" methods:
  System.out.println("First Artist Name: " +
                     firstArtist.getName());

  // Collection properties, such as the list of paintings, can be
  // obtained just like simple properties, by calling a
  // "get" method:
  List paintings = firstArtist.getPaintingArray();

  if(paintings.size() > 0) {
     Painting firstPainting = (Painting) paintings.get(0);
     System.out.println("1st Painting Name: " +
```

```
                        firstPainting.getName());
  }
}

// Catalog a new artist, Pablo Picasso, and his paintings
Artist picasso =
   (Artist) context.createAndRegisterNewObject(Artist.class);
picasso.setName("Pablo Picasso");

Painting selfPortrait =
   (Painting) context.createAndRegisterNewObject(Painting.class);
selfPortrait.setName("Self-portrait");
selfPortrait.setYear(new Integer(1907));

Painting theDream =
   (Painting) context.createAndRegisterNewObject(Painting.class);
theDream.setName("The Dream");
theDream.setYear(new Integer(1932));

// Set artist on both paintings; as a side effect this
// will automatically add these paintings to the Artist´s
// paintings collection.
selfPortrait.setToArtist(picasso);
theDream.setToArtist(picasso);

// Final step - commit changes.
// All three objects will be stored in the DB in one method call
context.commitChanges();
```

B.2.3. Baukästen

Expressions: Navigation des Objekt Graphen

Einer der Grundbausteine von Cayenne ist eine knappe und leserliche Ausdruckssprache für das Objekt und Datenbank Graph Traversal und SQL ähnliche Bedingungen. Es kommt an mehreren Stellen zum Einsatz wie der Query API und auch als eigenständiges Objekt-Graph Navigations Framework. Die Sprache ist selbsterklärend für jeden, der mit dem JavaBeans Properties Konzept vertraut ist. Zum Beispiel sind “name”, “paintingArray.toGallery.name” und “paintingArray.year > 2002” alles gültige Ausdrücke. Aufgrund der Tatsache, dass eventuell einige Spalten der Tabelle nicht im mapping der Objektschicht vorhanden sind und somit nicht als Objekteigenschaften eingesetzt werden können, erlaubt Cayenne die Generierung von Abfragen unter Verwendung von Datenbankspalten und Beziehungsnamen. Das Präfix ”db:” wird zur Unterscheidung von Datenbank und Java ”Properties” benutzt.

Wie bereits erwähnt, sind Expressions abstrakt und vom restlichen Framework unabhängig, wodurch sie hilfreich bei der Manipulation von JavaBeans sind, unabhängig davon, ob mit Objekten, die durch Cayenne gemapped wurden oder regulären Java Objekten gearbeitet wird.

- Auslesen von Objekt Eigenschaften

```
Expression exp = Expression.fromString("name");

Object object = ...;

// 'object' is any kind of Java object that has
// "getName()" method.
Object objectName = exp.evaluate(object);
```

- Filtern von Listen

```
// this particular expression does some DB-style
// regular expression matching on object properties
Expression filter = Expression.fromString("name like 'A%'");

List objects = ...;
List startWithA = filter.filterObjects(objects);
```

- Sortierung von Listen

```
// For ordering purposes, expressions are wrapped in
// an Ordering instance, that provides ordering direction
// in addition to the property specification and also
// implements java.util.Comparator
Ordering comparator = new Ordering("toArtist.name",
                                   Ordering.ASC);

List paintings = ...;

// sorts Paintings collection by artist name in
// ascending order
Collections.sort(objects, comparator);
```

Expressions sind dynamisch und können "named parameters" enthalten, die den Präfix "$" verwenden. Das ersetzen den Parameter geschieht mit Hilfe der Methode expWithParameters(..).

```
Expression exp = Expression.fromString
    ("toArtist.name like $namePattern or year = $year");
...
Map parameters = new HashMap();
parameters.put("namePattern", "X%");
```

```
parameters.put("year", new Integer(1954));

// 'anotherExp' will be: "toArtist.name like 'X%' or year = 1954"
Expression anotherExp = exp.expWithParameters(parameters);
```

Im Falle, dass einige der Parameter fehlen, ist die Methode intelligent genug um diese unbenutzten Bestandteile der Expression zu entfernen.

```
Expression exp = Expression.fromString
    ("toArtist.name like $namePattern or year = $year");
...
Map parameters = new HashMap();
parameters.put("namePattern", "X%");

// since "year = $year" part of the expression
// is automatically stripped, producing "toArtist.name like 'X%'"
Expression anotherExp = exp.expWithParameters(parameters);
```

Abfragen

Cayenne hat keinen eigenen SQL Dialekt und zwingt den Benutzer zu dessen Einsatz. Stattdessen wird eine Java Query API zur Verfügung gestellt. Der wohl am häufigsten eingesetzte Query type ist die SelectQuery um Objekte, sowie auch Roh Daten, aus der Datenbank auszulesen. Eine SelectQuery beinhaltet die Informationen für Cayenne, um die Datenbank zu finden, das SQL zu generieren und auszuführen und das Ergebnis in einem definierten Format zurück zu liefern. Die Anatomie von SelectQueries ist wie folgt

1. Query Root: Jede Abfrage hat ein "root" welches normalerweise eine Java Klasse ist. Dadurch wird Cayenne mitgeteilt welche source Tabelle oder view genutzt werden soll, welcher Type an Rückgabe Objekt erwartet wird und

letztlich welches DataNode für die Abfrage genutzt werden soll. Root ist der einzige teil der zwingend erforderlich ist.

```
SelectQuery query = new SelectQuery(Artist.class);
```

2. Query Qualifier: Der Qualifier spezifiziert welche Zeilen einer Abfrage geliefert werden sollen. In SQL werden diese in WHERE Klauseln gemapped.

```
Expression qualifier =
   Expression.fromString("toArtist.name like 'X%'");
SelectQuery query = new SelectQuery(Artist.class, qualifier);
```

3. Query Ordering: Werden in ORDER BY Klausens übersetzt und spezifizieren wie die Ausgabe sortiert werden soll.

```
SelectQuery query = new SelectQuery(Artist.class);
query.addOrdering("name", true);
query.addOrdering("dateOfBirth", true);
```

4. Query Prefetching: In dem fall das eine Beziehung zwischen Objekte existiert kann durch ein Prefetching erreicht werden das die "master" Objekte komplett mit den "detail" Objekten mit nur einer SQL Abfrage eingelesen werden anstatt für jedes "detail" Objekt eine eigene Abfrage absetzten zu müssen.

```
SelectQuery query = new SelectQuery(Painting.class);
query.addPrefetch("toArtist");
query.addPrefetch("toGallery");
```

5. Verschiedene Hinweise: Das SelectQuery hat eine Vielzahl von Eigenschaften mit denen eine zusätzliche Konfiguration erreicht werden kann.

```
SelectQuery query = new SelectQuery(Artist.class);

// resolve results page by page, 50 at a time
query.setPageSize(50);

// do not fetch more than 1000 records
query.setFetchLimit(1000);

// instead of Artist objects, fetch untyped Maps for each row
query.setFetchingDataRows(true);
```

Transaktionen

Cayenne bereit den Entwickler komplett von jeglicher Art von Transaktions-Management. DataContext.commitChanges() ist die einzige Methode, die normalerweise von Interesse ist. Es demarkiert einen Moment, indem alle Änderungen an Objekten innerhalb des DataContextes per JDBC an die Datenbank geschrieben werden.

Cayenne unterstützt auch explizite Transaktionen mit Hilfe der Transaction Klasse. Die Notwendigkeit, diese zu benutzten und den Cayenne eigenen Transaktionsmechanismus zu übergehen, sollte aber nur in den extrem seltensten Fällen nötig sein.

B.2.4. Weitere Features

Nun noch einige weitere Features, die erwähnenswert sind:

- Distributed Caching: Das DataDomain hat eine Objekt Caching Einrichtung das alle ihre DataContexte bedient und zudem auch über JVM Grenzen

hinweg verteilt werden kann. Die Verteilung über mehrere JVMs kann "out of the box" über JavaGroups und JMS realisiert werden.

- Batching und Sortierung: Cayenne setzt JDBC batching für alle DML Operationen ein, vorrausgesetzt der Adapter unterstützt dies. Dadurch wird ein signifikanter Performancegewinn erreicht bei der Verarbeitung von grossen Datenmengen.
- Vererbung: Cayenne unterstützt Vererbung von persistenten Klassen, solange sie die gleiche Basis-Tabelle verwenden.
- Optimistisches Locking: Der CayenneModeler ermöglicht es, eine Anzahl von Objekt Attributen und Beziehungen für ein optimistisches locking zu verwenden.
- "Lazy" Kollektionen und "hollow" Objekte: Cayenne ist "Lazy" im Auflösen von Beziehungen. Wenn ein Objekt eingelesen wird, ist es initial ein "hollow" Objekt, da alle Detail-Objekte noch nicht aus der Datenbank ausgelesen sind. Erst wenn diese Objekte per get angefordert werden, werden die Daten aus der Datenbank gelesen.

B.2.5. Cayenne vs. anderen ORM Frameworks

Nach einer Periode von Ablehnung, Ignoranz und Argwohn gegenüber der ORM Technologie ist sie heutzutage ein Mainstream in Java. So ist es nicht verwunderlich, dass es mehrere robuste Lösungen zur Auswahl gibt. Die Untersuchung einiger andere Frameworks wie EOF, TopLink und Hibernate hat gezeigt, dass alle ihre Aufgabe gut erfüllen, die gleichen Kern-Design Konzepte verfolgen und genügend Features besitzen, um jederzeit gegenüber JDBC-basierte DAOs bevorzugt zu werden. Leider ist ein Feature-für-Feature Vergleich der Frameworks nicht aussagekräftig. Am Ende ist es alles eine Frage des Geschmacks.

Zwei Sachen, die Cayenne befürworten, sind, der CayenneModeler und der DataContext. Durch die Integration eines GUI Modellers wird die Produktivität des

Benutzers gesteigert und die geistige Gesundheit erhalten. Keines der anderen Frameworks hat ein stimmiges und integriertes Modelling Tool.

Der DataContext tut für die Cayenne API, was der CayenneModeler für den Modelling Prozess getan hat - es macht es extrem einfach zu Benutzen.

Der DataContext ist "disconnected" und benutzt die Cayenne runtime für Konnektivität, dennoch kann es mit mehreren Datenbanken gleichzeitig arbeiten. Der DataNode stellt erst dann eine connection zur Verfügung, wenn die eigendliche Operation gestartet wird und weiss, wie Daten commited und die connection wieder geschlossen werden muss. Der DataContext ist zusammen mit all seinen Objekten serialisierbar und kann mehrere Transaktionen überbrücken. Auf der Code-Ebene ist kein Transaktions-Management erforderlich. All dies macht den DataContext zum idealen session-level Persistenten Objekt Container.