

Technische Universität  
München

Fakultät für Informatik

Forschungs- und Lehrereinheit Informatik IV

## Systementwicklungsprojekt

### Migration von MS-Access zu webbasierten Lösungen

#### **Aufgabensteller**

Prof. Dr. M. Broy

#### **Teilprojekt**

Implementierung der Anwendung mit Hilfe von Turbine

#### **Bearbeiter**

Guo Sheng ([sheng@in.tum.de](mailto:sheng@in.tum.de))

Rong Xie ([xier@in.tum.de](mailto:xier@in.tum.de))

#### **Teilprojekt**

Synchronisationstool MS Access – PostgreSQL

#### **Bearbeiter**

Andreas Greimel ([andreas.greimel@web.de](mailto:andreas.greimel@web.de))

#### **Betreuer:**

Stefan Wagner ([wagnerst@in.tum.de](mailto:wagnerst@in.tum.de))

Martin Huber ([huber@cdtm.de](mailto:huber@cdtm.de))

|        |  |    |
|--------|--|----|
| 1      | Einleitung.....                                  | 3  |
| 2      | Anwendung.....                                   | 4  |
| 2.1    | Grundlagen .....                                 | 4  |
| 2.1.1  | Entwicklungsvorgaben.....                        | 4  |
| 2.1.2  | Komponenten .....                                | 4  |
| 2.1.3  | Turbine .....                                    | 5  |
| 2.1.4  | Services.....                                    | 8  |
| 2.1.5  | Pull MVC Model.....                              | 8  |
| 2.1.6  | Sicherheitssystem.....                           | 9  |
| 2.1.7  | Torque .....                                     | 10 |
| 2.1.8  | Velocity .....                                   | 10 |
| 2.1.9  | Avalon.....                                      | 11 |
| 2.1.10 | Ant.....   | 11 |
| 2.2    | Design .....                                     | 13 |
| 2.2.1  | Bestandteile einer Komponente.....               | 13 |
| 2.2.2  | Entwicklungsumgebung.....                        | 13 |
| 2.2.3  | Gesamter Überblick .....                         | 14 |
| 2.2.4  | Detaillierte Strukturen .....                    | 16 |
| 2.3    | Implementierung.....                             | 32 |
| 2.3.1  | Die Entwicklungsumgebung.....                    | 32 |
| 2.3.2  | Verwaltungsbereich .....                         | 32 |
| 2.3.3  | Ant Build-Datei.....                             | 34 |
| 2.3.4  | Oberfläche .....                                 | 35 |
| 2.3.5  | Basic Data Management (bad und badGUI) .....     | 35 |
| 2.3.6  | Order Management (oma und omaGUI) .....          | 41 |
| 2.3.7  | Generierung von Charts (chart) .....             | 42 |
| 2.3.8  | Generierung von Berichten (report) .....         | 43 |
| 2.4    | Ausblicke.....                                   | 45 |
| 3      | Synchronisation .....                            | 46 |
| 3.1    | Situation/Aufgabenstellung.....                  | 46 |
| 3.2    | Realisierung .....                               | 46 |
| 3.2.1  | Die Datenbankstruktur .....                      | 46 |
| 3.2.2  | Mapping.....                                     | 47 |
| 3.2.3  | Klassen.....                                     | 50 |
| 3.2.4  | Einschränkungen während parallelen Betriebs..... | 71 |
| 3.3    | Probleme bei der Synchronisation.....            | 72 |
| 3.4    | Fazit/Bewertung .....                            | 72 |
| 4      | Zusammenfassung .....                            | 75 |

# 1 Einleitung

In diesem Systementwicklungsprojekt (SEP) sollen die Möglichkeiten und die Grenzen einer Systemersetzung ermittelt werden. Das alte System ist eine Access-Lösung, die Nachteile wie fehlende Mehrbenutzerfähigkeit, Sicherheit und Lizenzgebühren aufweist, aber sehr einfach zu benutzen ist. Außerdem erreicht Access schnell Kapazitätsgrenzen und es muss sich nach einer anderen Lösung umgeschaut werden, meist ein neues, besseres System. Das neue System sollte eine Datenbank sein, die die aufgezählten Accessspezifischen Nachteile nicht mehr besitzt, beispielsweise PostgreSQL oder MySQL. Die Bedienung und Verwaltung der Daten könnte über eine Web Schnittstelle stattfinden, so dass jeder Mitarbeiter bequem über das Intranet Zugriff auf das neue System hat. Wichtige Voraussetzung hierbei ist oft, dass es sich um Open Source, also nicht kommerzielle Produkte handeln soll.

Das Ausmaß der Systemersetzung wird anhand eines konkreten Falles untersucht. Ein Medien Verlag besitzt eine Access Datenbank mit sämtlichen Kundendaten und Auftragsdaten. Diese Lösung wird ersetzt durch eine PostgreSQL Datenbank und eine Web Schnittstelle, über die die Daten verwaltet werden können. Das Projekt wurde in Zusammenarbeit mit der Firma ‚gogol medien‘ durchgeführt.

Es wurden die Mittel und die Vorgehensweise festgesetzt. Als Backend Datenbank wurde sich für das Open Source Produkt PostgreSQL entschieden. Die Datenbankstruktur wurde von gogol medien festgelegt. Der Webzugriff wird mit dem Apache Projekt Turbine realisiert, ein nicht kommerzielles Werkzeug zur Entwicklung von Webanwendungen.

Das neue System sollte inkrementell eingeführt werden, d.h. dass beide Systeme, Access und PostgreSQL eine zeitlang parallel laufen werden. Um die Daten vom alten auf das neue System zu übernehmen und während des parallelen Betriebs aktuell zu halten wird noch ein Synchronisationstool entwickelt. Dieses Tool wird mit PHP4 entwickelt.

Der ersten Teil der Ausarbeitung beschreibt kurz die benutzen Werkzeuge zur Entwicklung der Webanbindung. Dann wird die Realisierung beschrieben. Der zweite Teil geht ausführlich auf die Entwicklung des Synchronisationstools ein.

## 2 Anwendung

### 2.1 Grundlagen

#### 2.1.1 Entwicklungsvorgaben

Da die Aufgabensteller eine Entwicklungsumgebung schon zur Verfügung gestellt haben, sollen die neu zu entwickelnden Komponenten unter dieser Umgebung realisiert werden. Die Umgebung ist mit der Programmiersprache Java und dem Framework Turbine(vgl. Abschnitt 3.1.3) implementiert, außerdem sind Torque (vgl. Abschnitt 3.1.7) und Avalon (vgl. Abschnitt 3.1.9) auch eingebunden. Soweit möglich sollen aus sicherheits- kostentechnischen Gründen für die Entwicklung nur Werkzeuge aus dem Open Source-Bereich benutzt werden, die eine kommerzielle Nutzung nicht ausschließen.

#### 2.1.2 Komponenten

Komponenten sind in [19] folgendermaßen definiert:

*“Das Wort "Komponente" stellt ein physisches Stück Programmcode (z.B. Quellcode, Binärcode, DLL, ausführbares Programm) dar. Komponenten ähneln Paketen. Sie gruppieren und gliedern eine Menge einzelner Elemente, definieren Grenzen und können über Schnittstellen verfügen”*

Eine Komponente kann am besten mit einem Baustein verglichen werden, dessen Inneres verborgen ist. Man erkennt jedoch, dass er mit anderen Bausteinen über seine Verbinder zusammengefügt werden kann. Bei Kombination passender Bausteine erhält man ein Gebilde, das einen bestimmten Zweck erfüllt.

Eine Komponente kann, neben Schnittstellen zu anderen Komponenten, auch über Schnittstellen zu den Benutzern verfügen. Diese Benutzer-Schnittstellen werden bei webbasierten Anwendungen meist von HTML-Seiten in Verbindung mit Grafiken dargestellt. Sie stellen das *“graphical user interface”* (GUI) dar und sind für die Interaktion mit dem Benutzer zuständig.

Bei der Entwicklung von webbasierten Komponenten spielt der Sicherheitsaspekt eine wichtige Rolle. Da es im Normalfall keine automatische Authentifizierung von Benutzern möglich ist, könnte jeder, der über einen Zugriff auf das entsprechende Netzwerk verfügt, die Anwendung aufrufen. Deshalb ist es sinnvoll, eine Unterscheidung zwischen öffentlichen und nichtöffentlichen Schnittstellen bei den Komponenten zu definieren. Vor der Benutzung einer nichtöffentlichen Schnittstelle ist eine Authentifizierung notwendig. Durch eine fein gegliederte Rechtestruktur ist es des weiteren möglich innerhalb nichtöffentlicher Bereiche weitere Restriktionen vorzunehmen.

## 2.1.3 Turbine

Für die Entwicklung Web-Basierter Anwendungen sind heutzutage viele Projekte, Sprachen und Werkzeuge erschienen. Speziell im Java-Umfeld sind neben den grundlegenden Techniken wie JSP und Servlets komplexe Web-Frameworks entstanden, die dem Entwickler nach der Einarbeitung eine Menge hilfreicher Komponenten zur Verfügung stellen. Eines dieser Frameworks ist das Open-Source-Projekt Turbine der Apache Software Foundation.<sup>1</sup>

Zu Beginn gestaltet sich die Umsetzung einer Anwendung mit dem Turbine-Framework komplex und erfordert eine intensive Einarbeitungszeit, bevor der Entwickler von den zur Verfügung gestellten Techniken profitieren kann. Dann ist aber das Framework ein mächtiges Werkzeug zur Entwicklung web-basierter Anwendungen mit Datenbankanbindung.

Die prinzipielle Funktionsweise der mit Turbine erstellten Anwendungen entspricht dem MVC-Paradigma (Model, View, Controller). Dieses stammt ursprünglich aus dem Umfeld der Programmiersprache Smalltalk und ist mittlerweile Standard für die Entwicklung der Web-Basierten Anwendungen.

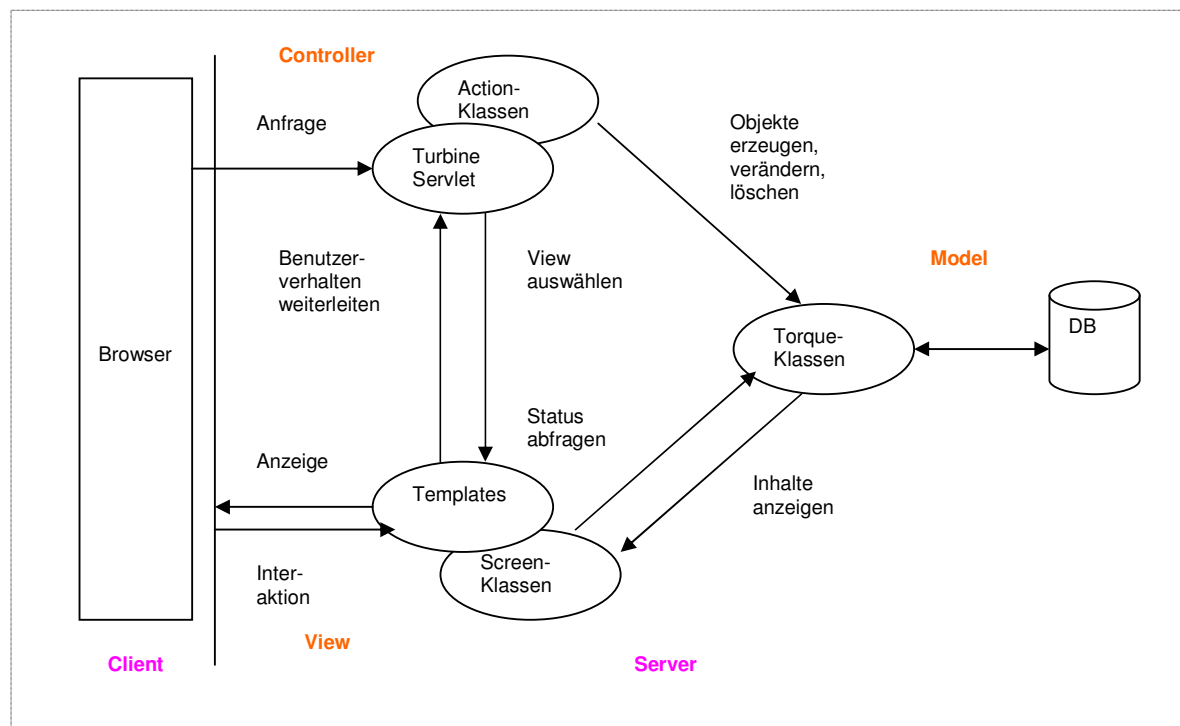


Abbildung 2-1 Struktur von dem Turbine Framework<sup>2</sup>

Das MVC-Paradigma (siehe [11]) unterteilt eine Anwendung in drei Komponenten mit unterschiedlichen Zuständigkeiten. Die Model-Komponente definiert die Geschäftslogik einer Anwendung und ermöglicht im Allgemeinen den Zugriff auf

<sup>1</sup> siehe <http://jakarta.apache.org/turbine/>

<sup>2</sup> IX Magazin für professionelle Informationstechnik November 2003

deren Daten. Hierzu stellt sie Funktionen zum Speichern, Verändern und Löschen bereit. Die Hauptaufgabe der View-Komponente ist es, die Anwendungsdaten des Models für den Benutzer geeignet darzustellen. Darüber hinaus ist diese Komponente dafür zuständig, die Benutzerinteraktion an die dritte Komponente, den Controller, weiterzuleiten. Dieser wiederum verändert den Zustand des Models, in dem er auf die Interaktion reagiert. Zusätzlich wählt der Controller die jeweilige View-Komponente zur Darstellung der Daten aus.

Java-Entwickler sprechen bei einer Umsetzung des MVC-Paradigmas, häufig von der Model-2-Architektur, wobei ein Servlet dem Controller entspricht, JSP-Seiten die Views und Java-Beans das Model repräsentieren. Turbine setzt auf diese Model-2-Architektur auf, erweitert sie allerdings um einige Komponenten – daher die Bezeichnung Model-2+1-Architektur (siehe [9]). Zu den Erweiterungen zählen beispielsweise Action-Klassen, die dem Controller ergänzen. Dabei übernimmt der Controller – das so genannte Turbine-Servlet – die allgemeine Ablaufkontrolle, und behandelt die einzelnen Benutzeranfragen. Die Action-Klassen sind anwendungsspezifische Erweiterungen, die das Turbine-Servlet aufruft.

Im Rahmen dieses Projektes kommt Turbine in der Version 2.3 zum Einsatz. Die nachfolgenden Unterabschnitte erläutern deren grundlegende Technik.

## Turbine-Module

Der Kern von Turbine besteht aus fünf Modulen, dem *Action*-, *Layout*-, *Navigation*-, *Screen*- und dem *Page*-Modul (vgl. Abb. 3-2).

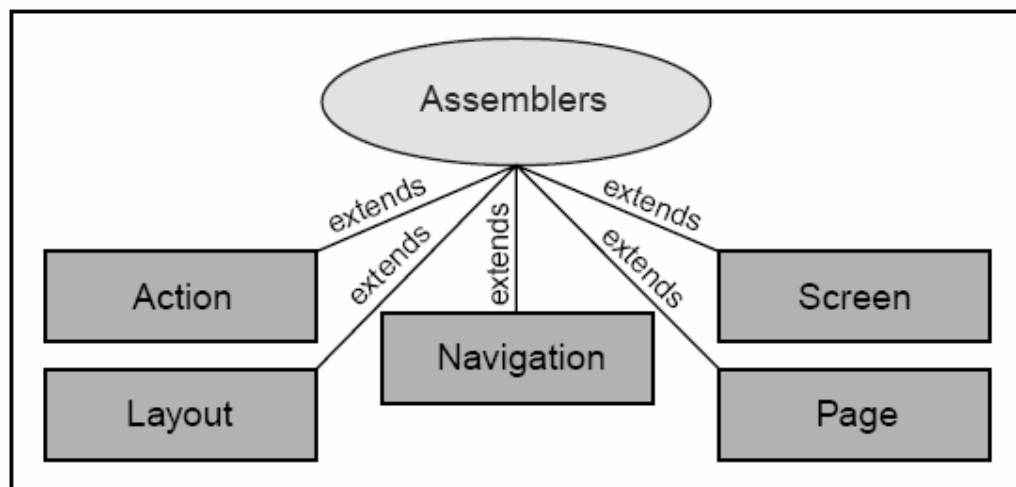


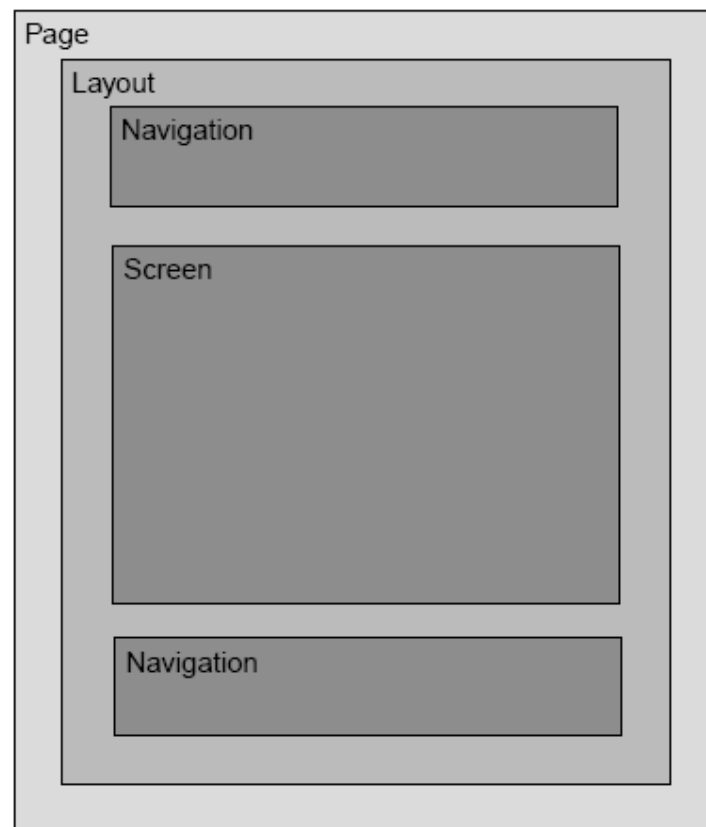
Abbildung 2-2 Die Module von Turbine<sup>1</sup>

Das *Action*-Modul ist zuständig für die Behandlung von Ereignissen, vergleichbar mit dem ereignisbasierten Programmieren. Im Allgemeinen kann man sagen, dass eine *Action* eine wieder verwendbare Einheit darstellt, die eine bestimmte Aufgabe erfüllt. Sie kann in mehrere, selbstdefinierbare Ereignisse unterteilt werden. Wird bei dem Aufruf einer *Action* kein Ereignis angegeben, so wird defaultmäßig das Ereignis

<sup>1</sup> siehe <http://jakarta.apache.org/turbine/turbine-2.3/fsd.html>

*doPerform* ausgeführt. Eine typische Aufgabe für eine *Action* ist beispielsweise die Verarbeitung von Daten eines HTML-Formulars oder auch der Logoutvorgang.

Bei Turbine ist eine Webseite in die Bereiche *Navigation* und *Screen* aufgeteilt. Das *Layout*-Modul definiert und organisiert diese Aufteilung. Im *Screen*-Bereich wird der eigentliche Inhalt einer Seite dargestellt. Pro *Layout* kann jeweils nur ein *Screen*-Bereich festgelegt werden. *Navigation*-Bereiche können dagegen beliebig viele spezifiziert werden, z.B. im oberen und unteren Teil der Seite.

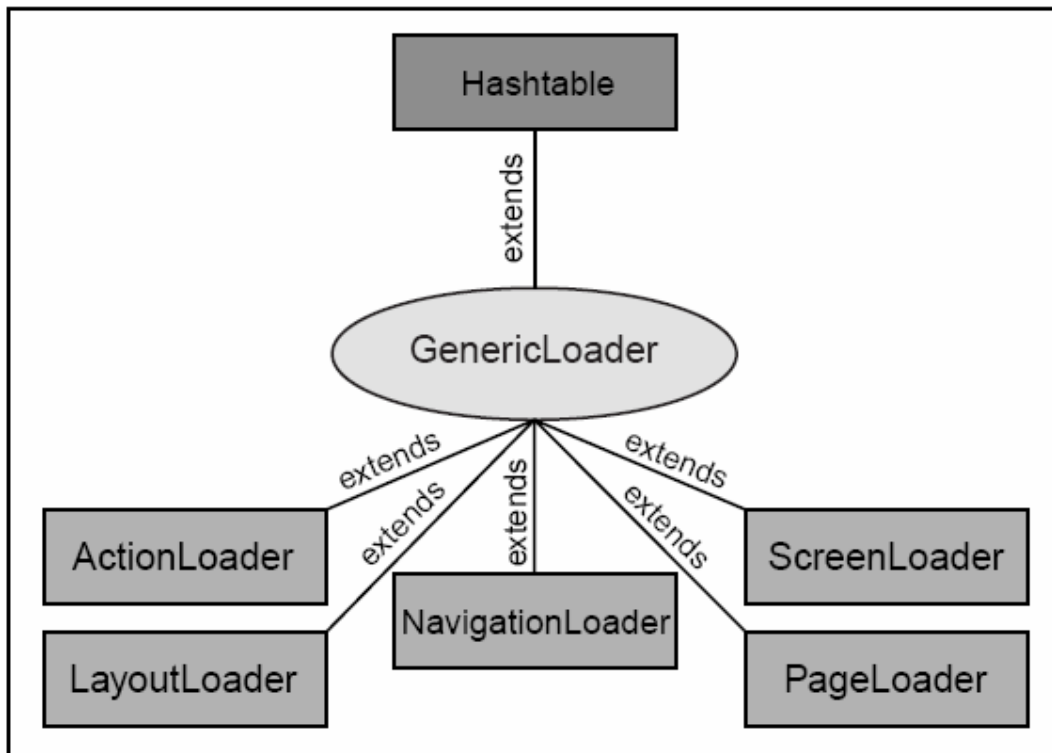


**Abbildung 2-3 Die Modulverschachtelung von Turbine<sup>1</sup>**

Die Generierung einer Webseite erfolgt schrittweise durch die Ausführung der einzelnen Module, wobei beim *Page*-Modul gestartet wird. Dies überprüft ob eine *Action* und ein dazugehöriges Ereignis definiert ist. Wenn ja, wird das definierte Ereignis, bzw. das Default-Ereignis ausgeführt. Das *Page*-Modul ruft dann anschließend das *Layout*-Modul auf. Das *Layout*-Modul wiederum ruft das *Screen*- und das *Navigation*-Modul auf (vgl. Abb. 3-3). Innerhalb einer *Action* besteht die Möglichkeit die darzustellende Seite zu überschreiben. Dies ermöglicht z.B. die erneute Darstellung eines HTML-Formulars, falls die Validierung der Daten fehlgeschlagen ist.

Die einzelnen Module werden über so genannte *Modul-Loader* dynamisch geladen (vgl. Abb. 3-4). Optional besteht die Möglichkeit die Module im Hauptspeicher vorzuhalten (*Modul-Caching*), um somit eine schnellere Ladezeit zu gewährleisten.

<sup>1</sup> siehe <http://jakarta.apache.org/turbine/turbine-2.3/fsd.html>



**Abbildung 2-4 Die Modul-Loader von Turbine<sup>1</sup>**

Die *Modul-Loader* benutzen zum Auffinden der einzelnen Module einen so genannten *Loader Classpath*, der über die Konfigurationsdatei *TurbineResources.properties* spezifiziert werden kann. Ähnlich dem *CLASSPATH* in Java, definiert dieser an welchen Stellen sich die Module befinden können.

## 2.1.4 Services

Die meisten Funktionalitäten werden in Turbine durch *Services* zur Verfügung gestellt. Dies sind so genannte *Singletons*, d.h. es gibt innerhalb der Anwendung nur eine Objekt-Instanz. Daher werden benötigte System-Ressourcen, wie z. B. Arbeitsspeicher, nur einmal verbraucht. Jedoch muss daher auch beachtet werden, dass der interne Zustand eines Services für alle anfragenden Clients gleich ist (Stichwort "*threadsafe programming*").

Turbine Services werden über die Konfigurationsdatei *TurbineResources.properties* eingebunden und konfiguriert. Daher ist es relativ einfach eigene Services zu entwickeln und sie in das Gesamtsystem zu integrieren.<sup>2</sup>

## 2.1.5 Pull MVC Model

<sup>1</sup> siehe <http://jakarta.apache.org/turbine/turbine-2.3/fsd.html>

<sup>2</sup> siehe <http://jakarta.apache.org/turbine/turbine-2.3/services/index.html>

Bei der Standard-MVC-Umsetzung von Turbine ist jedem Template eine *Screen*-Klasse zugeordnet. (siehe [12]) Dieser instanziiert die benötigten Objekte und stellt sie dem Template zur Darstellung zur Verfügung. Dadurch kann das *“look and feel”* der entsprechenden Webseite, also das Template, ohne jegliche Java-Kenntnisse modifiziert werden. Diese Architektur ist sehr einfach zu verstehen und zu implementieren. Die feste Zusammengehörigkeit von Template und Screen erweist sich jedoch oft als Nachteil. Ein Web-Designer kann dadurch nicht ohne weiteres Informationen von einem Template zu einem anderen Template verschieben, da hierfür eine Modifikation der entsprechenden Screen-Klasse notwendig wäre. In Turbine gibt es jedoch die so genannten *“Pull Tools”*. Pull Tools sind spezielle Java-Klassen, die eine gewisse Funktionalität zur Verfügung stellen, ähnlich den Turbine Services (siehe Abschnitt 3.1.4). Sie sind jedoch, im Unterschied zu den Services, direkt in den Templates zugänglich und zwar in jedem Template innerhalb der Anwendung. Des weiteren werden bei den Pull Tools folgende Möglichkeiten der Objekt-Instanziierung unterschieden:

- *global* - Es existiert nur eine Instanz (analog zu den Services).
- *request* - Für jeden Request existiert eine eigene Instanz.
- *session* - Für jede Benutzer-Session existiert eine eigene Instanz.
- *persistent* - Für jede Benutzer-Session existiert eine eigene Instanz. Der Zustand dieser Instanz wird jedoch beim Abmeldevorgang des Benutzers persistent gespeichert. Er wird bei der nächsten Anmeldung wiederhergestellt und steht somit dem Benutzer wieder zur Verfügung.

Daten, die über ein Pull-Tool verfügbar sind, können nun in beliebigen Templates dargestellt werden. Dies ermöglicht eine sehr schnelle und einfache Änderbarkeit des Workflows einer Anwendung und führt zu einer noch größeren Unabhängigkeit von Web-Designer und Java-Entwickler.

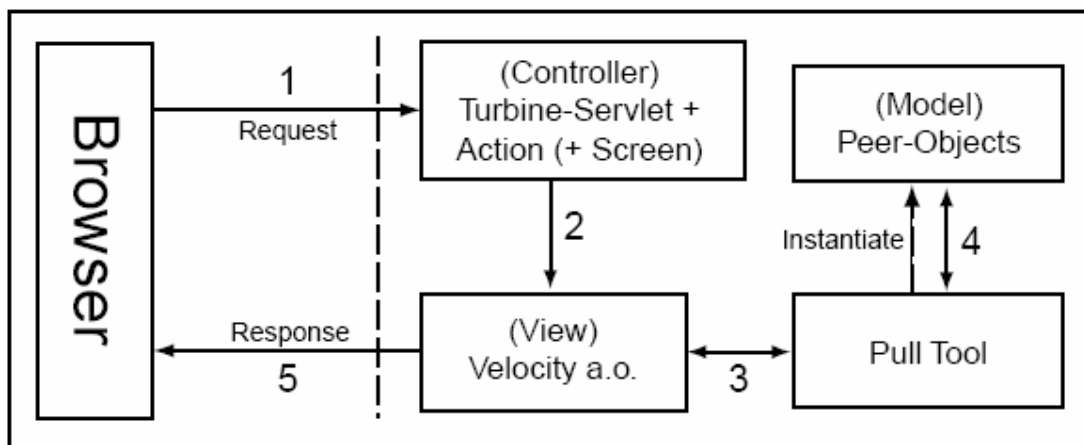


Abbildung 2-5 Das Pull MVC-Model in Turbine

## 2.1.6 Sicherheitssystem

In vielen Anwendungen ist es notwendig die zur Verfügung gestellten Funktionalitäten je nach Benutzerrechten einzuschränken.

Turbine unterstützt dies durch ein sehr einfaches aber dennoch mächtiges Sicherheitssystem. Es basiert auf so genannten *Access Control Lists*. Dabei wird zu dem aktuellen Benutzer eine Liste von Berechtigungen bestimmt, die über einfache Schnittstellen abgefragt werden können. Die Berechtigungen werden einem Benutzer indirekt über eine oder mehrere Rollen zugewiesen. Eine Rolle ist eine Sammlung von Berechtigungen. Des Weiteren besteht die Möglichkeit Gruppen festzulegen, wobei ein Benutzer in unterschiedlichen Gruppen unterschiedliche Rollen haben kann. Somit kann ein Benutzer beispielsweise in einer Projektgruppe A der Projektleiter und in einer Projektgruppe B ein Programmierer sein.

## 2.1.7 Torque

Hilfe bei der Erstellung des Modells bietet das Tool Torque an. Torque war ursprünglich in Turbine integriert, und ist seit der Turbine Version 2.2 ein selbstständiges Werkzeug. (siehe [14]) Torque ist eine objektorientierte Schnittstelle für eine relationale Datenbank. Die Geschäftslogik setzen Klassen um, die die generierten Torque Klassen aggregieren. Um die Torque Klassen generieren zu können, muss man vorher die Datenbankdefinitionen, sowie der Name der Datenbank, die Tabellendefinitionen, usw., in einer XML Datei schreiben. Die Struktur der XML Datei ist nach einer DTD Datei festgelegt.<sup>1</sup> Torque nützt die Datei dann als Grundlage, bildet danach mit ANT (siehe Abschnitt 3.1.10) die entsprechenden Peer Klassen und eine SQL Datei. Mit Hilfe dieser SQL Datei kann man die gewünschte Datenbank und alle Tabellen erzeugen. Durch die Peer Klassen ist der Zugriff auf die einzelnen Tabellen und die einzelnen Einträge in der Datenbank möglich. Die Peer Klassen bieten zusätzlich noch Methoden, um die Einträge in einer Tabelle zu löschen, ändern oder neue Einträge einzufügen.<sup>2</sup>

## 2.1.8 Velocity

Für die Erstellung der View-Komponenten empfehlen die Turbine Entwickler ihre eigene Velocity-Technik. Die zugrund liegende Velocity-Template-Language (VTL) besitzt im Vergleich zu anderen Programmiersprachen (Java, VB, etc.), die in Template-Techniken wie JSP oder ASP Verwendung finden, nur wenige Sprachkonstrukte. (siehe [16]) Dies soll vermeiden, dass Templates die komplette Geschäftslogik enthalten. Das dynamische Verhalten – die Action – eines Formulars definiert der in Turbine integrierte Pull-Service (siehe Abschnitt 3.1.5) *link*<sup>3</sup>. Ein solcher Service macht im Turbine Framework für jedes Template häufig gebrauchte Daten verfügbar. *Link* dient im Speziellen dazu, auf einfache Art Turbine-konforme Verweise zu erstellen.

Die Velocity-Template-Language (VTL) hat Vielzahl von Einsatzmöglichkeiten. Denkbar wäre beispielsweise die kontextabhängige Generierung von SQL-Code,

---

<sup>1</sup> siehe [http://db.apache.org/torque/dtd/database\\_3\\_1.dtd](http://db.apache.org/torque/dtd/database_3_1.dtd)

<sup>2</sup> siehe <http://db.apache.org/torque/>

<sup>3</sup> *link* ist ein Objekt aus VTL, das die Aufgabe hat, die Verweise zu einer bestimmten Seite zu erstellen.

PostScript oder Java Quellcode ausgehend von entsprechenden Templates. Wenn sie aber mit Turbine zusammenarbeitet, dann ist die Anwendung nach einem „echten“ MVC-Model entwickelt.<sup>1</sup>

### 2.1.9 Avalon

Das Apache Avalon Projekt besteht aus einer Menge von Produkten und Subprojekten, die verschiedene Services und das Management von Komponenten unterstützt. Außerdem bietet Avalon noch eine komplette Plattform für die Implementierung von Komponenten an, mit deren Hilfe die Implementierung erleichtert wird. (siehe [20]) Die wesentlichen Vorteile von Avalon sind:

- Eine stabile Entwicklungsumgebung von Komponenten (jetzt nur mit der Programmiersprache Java)
- Management von Lebenszyklus<sup>2</sup> der Komponenten
- Management der Komponentenkonfiguration
- „easy-to-use“ API
- Management der Abhängigkeiten von Services
- Geeignet für standalone, J2EE und Web Umgebungen

Im Rahmen dieses Projektes kommt nur der Avalon Komponentenservice zum Einsatz. Der Avalon Komponentenservice lädt zuerst die externen Module, die die Schnittstellen vom Lebenszyklus von Avalon<sup>2</sup> implementieren, danach werden die Module von dem Komponentenservice verwaltet. Eine der bisherigen Implementierungen ist eine Torque (siehe Abschnitt 3.1.7) Implementierung.

### 2.1.10 Ant

*Ant* ist ein Java-basiertes *Build Tool*. Dies sind, neben dem Kompilieren des Quellcodes, auch das Zusammenstellen von Programmpaketen, das Durchführen von Tests und die Erzeugung von Code-Dokumentationen.

Ant ist vergleichbar mit dem in der Unix-Welt weit bekannten *make*. Zur Beschreibung aller projektbezogenen Abhängigkeiten verwendet es ebenfalls eine *Build-Datei*. Anders als *make*, dessen Syntax sehr gewöhnungsbedürftig ist, nutzt Ant jedoch ein leicht lesbares XML-Format. In einer Build-Datei werden *Targets* (Ziele) definiert. Jedes Target führt eine oder mehrere Aufgaben (*Tasks*) aus. Eine typische Aufgabe wäre, wie oben schon genannt, das Kompilieren des Quellcodes.

---

<sup>1</sup> siehe <http://jakarta.apache.org/velocity/>

<sup>2</sup> siehe <http://avalon.apache.org/>

Ant entfaltet seine volle Leistungsfähigkeit insbesondere bei größeren Projekten. Hier ist besonders die Integrationsfähigkeit in renommierte Entwicklungsumgebungen, wie z. B. *JBuilder*, *Eclipse* und *NetBeans*, zu erwähnen.<sup>1</sup>

---

<sup>1</sup> siehe <http://ant.apache.org/>

## 2.2 Design

In diesem Kapitel werden die Strukturen bzw. Vorgaben für die spätere Implementierung definiert. Dabei muss im Wesentlichen zunächst einmal festgelegt werden, was denn zu einer Komponente gehört. Dann gilt es die neuen Komponenten für die Entwicklungsumgebung zu definieren. Anschließend werden Anforderungen an die einzelnen Bereiche formuliert.

### 2.2.1 Bestandteile einer Komponente

Wie bereits ansatzweise aus Abschnitt 3.1.2 ersichtlich wird, könnte man Komponenten in drei wesentliche Bereiche untergliedern.

- Templates – (optionaler Bereich, wenn eine Komponente nur Geschäftslogik enthält, dann ist keine Template erforderlich) Die Templates stellen im Zusammenspiel mit einer Template-Engine (beispielsweise *Velocity*, vgl. Abschnitt 3.1.8) die Benutzerschnittstelle dar. Sie verknüpfen statische Informationen zur Darstellungsweise, meist HTML-Anweisungen in Verknüpfung mit Grafiken, mit dynamischen Elementen der Anwendung und führen für den Benutzer transparent die Zugriffe auf die eigentlichen Anwendungsschnittstellen durch.
- Konfigurationsbereich - Es können in diesem Bereich komponentenspezifische Konfigurationsdateien platziert werden, mit deren Hilfe Einstellungen, die das Verhalten einer Komponente beeinflussen, vorgenommen werden können.
- Ausführungslogik - In diesem Bereich werden die eigentlichen Arbeiten der jeweiligen Komponente erledigt (z.B. Berechnungen, Datenbeschaffung und -aufbereitung usw.). Alle zu einer Komponente gehörenden Java-Klassen sind diesem Bereich zuzuordnen.

### 2.2.2 Entwicklungsumgebung

Wie bereits im Abschnitt 3.1.1 erwähnt wurde, ist die Entwicklungsumgebung für das Projekt schon festgelegt, und das Projekt soll unter dieser Umgebung die folgenden neuen Komponenten realisieren:

- Eine Verwaltungskomponente, die für das Management von Kontakten zuständig ist, wobei ‚Management‘ heißt, dass es durch diese Komponente das Anzeigen von den bereits existierenden Kontakten, das Einfügen von neuen Kontakten, das Verändern von Kontakten und das Löschen von Kontakten möglich sind. Das Löschen hier heißt nicht, dass ein Datensatz wirklich aus der Datenbank entfernt wird, sondern wird dieser Datensatz bloß bei der Anzeige ausgeblendet.

- Eine Verwaltungskomponente, die für das Management von Aufträgen zuständig ist, wobei ‚Management‘ heißt, dass es durch diese Komponente das Anzeigen von den bereits existierenden Aufträgen, das Einfügen von neuen Aufträgen und das Verändern von Aufträgen möglich sind.
- Eine universale Komponente, die als Eingabe ein paar Datensätze bekommt, und als Ausgabe einen durch die Daten generierten Chart zurückgibt.
- Eine Komponente, die als Eingabe alle Informationen aus einem Auftrag bekommt, und dadurch einen Bericht generiert. Der Bericht soll möglicherweise auch als PDF gespeichert werden können.

### 2.2.3 Gesamter Überblick

Bevor die einzelnen Komponenten detailliert beschrieben werden, ist hier zunächst ein Überblick über das gesamte System. (Abbildung 3-6)

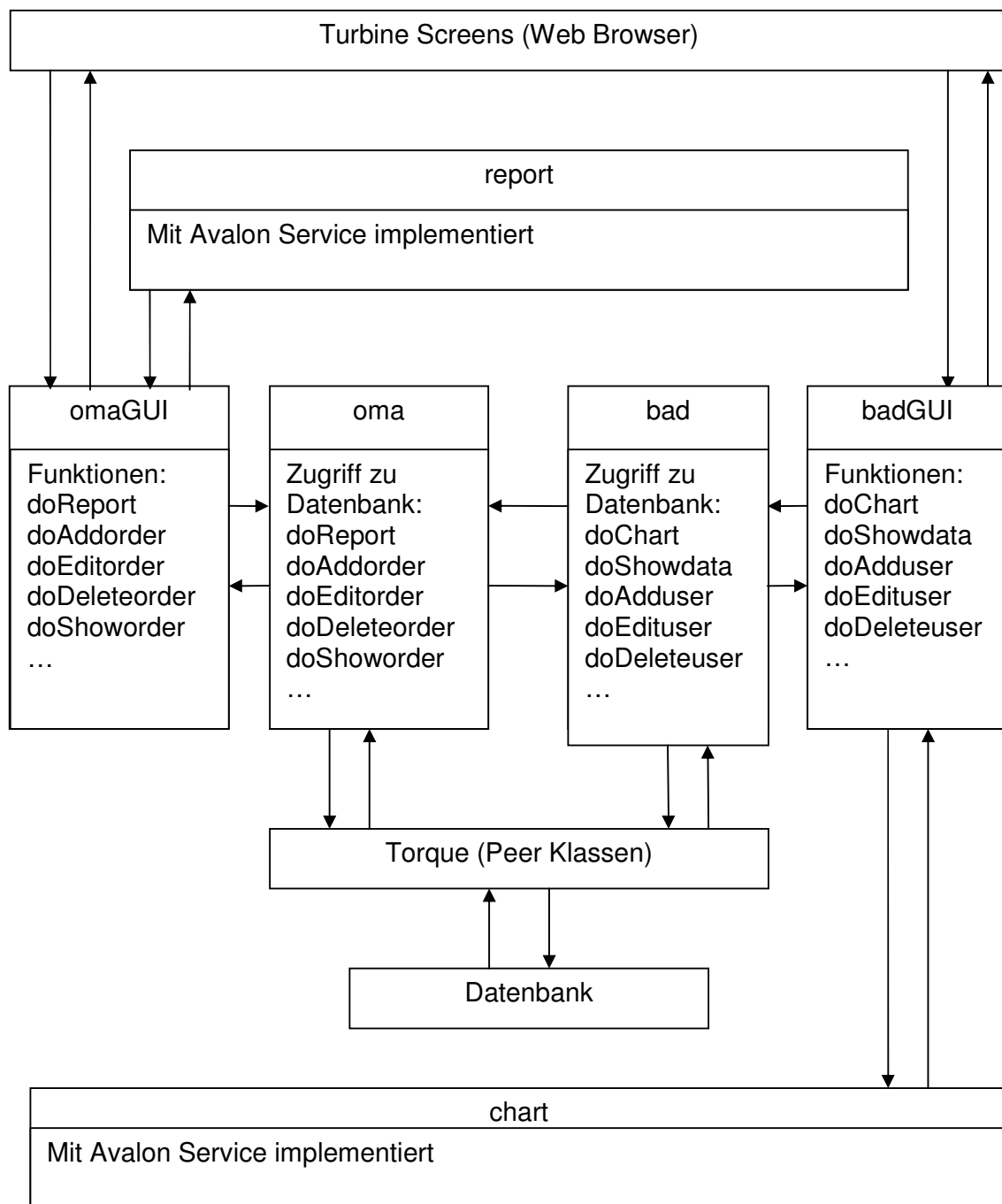
Der Teil „Turbine Screens“ arbeitet mit einem Web Browser<sup>1</sup> zusammen, um die Daten und Ergebnissen anzuzeigen. Hier ist auch die einzige Schnittstelle zu den Benutzern, und repräsentiert die View-Komponente nach dem MVC-Paradigma. (siehe Abschnitt 3.1.3)

Mit Hilfe von den von Torque automatisch generierten Peer-Klassen ist der direkte Zugriff zu der Datenbank möglich. Für die Generierung von den Peer-Klassen wird zunächst eine XML-Schema Datei manuell erstellt, in der die Grundinformationen von der Datenbank definiert sind. Ganz wichtig bei der Entwurfsphase ist, dass die schon in der Datenbank existierenden Tabellen in zwei Gruppen verteilt werden müssen, da nach den Anforderungen zwei verschiedene Komponenten implementiert werden. Die Komponente „bad“ (steht kurz für basic data) in der Abbildung 3-6 repräsentiert die Benutzerverwaltung. Sie hat den Zugriff zu der Datenbank, und kann die Daten direkt aus der Datenbank holen, speichert neue Dateneinträge in der Datenbank ab, ändert einen Dateneintrag oder löscht einen spezifizierten Dateneintrag. Sie repräsentiert die Model-Komponente nach dem MVC-Paradigma. (siehe Abschnitt 3.1.3) Analog ist es auch bei der Komponente „oma“ (steht kurz für order management), bloß ist diese Komponente für die Auftragsverwaltung zuständig statt für die Benutzerverwaltung. Die beiden Komponenten sind aber nicht isoliert, d.h. sie kommunizieren in den Fällen, falls z.B. eine Funktion bei „bad“ die Auftragsdaten benötigt, oder eine andere Funktion bei „oma“ die Benutzerdaten benötigt. In den beiden Fällen wird sich eine Komponente an die andere Komponente wenden, um die benötigten Daten aus der Datenbank zu holen und übergeben sie an die beiden GUI-Komponenten. Die GUI-Komponenten kann man sich so vorstellen, dass sie eine Verbindung (oder eine Brücke) zwischen der Web-Browser (oder der Benutzerinteraktion) und den Komponenten, die den Zugriff direkt zu der Datenbank haben, darstellt. Sie entsprechen der Controller-Komponente nach dem MVC-Paradigma. (siehe Abschnitt 3.1.3) Nach der Namenskonvention von Avalon Service

---

<sup>1</sup> Es ist geplant, dass das System sowohl unter Internet Explorer, als auch unter Mozilla funktionieren soll.

heißen die Methoden in den beiden GUI-Komponenten genauso wie bei den Komponenten „bad“ und „oma“.



**Abbildung 2-6 Gesamter Überblick**

Die Komponenten „chart“ und „report“ sind zwei Funktionskomponenten, die als Eingabe ein paar Daten bekommen, und rechnet dadurch die Datensätze für einen Chart bzw. einen Bericht, und als Ausgabe wird das Ergebnis in eine gegebene Datei

geschrieben. Diese Datei wird dann durch die View-Komponente in der Web-Browser angezeigt.

## 2.2.4 Detaillierte Strukturen

Nachdem die grobe Struktur des Systems klargemacht wird, wird es in diesem Abschnitt die detaillierten Strukturen der einzelnen Komponenten erklärt.

### 2.2.4.1 Basic Data → bad

„bad“ ist eine von den beiden Komponenten, die den Zugriff zu der Datenbank haben. Sie bietet die Methoden zum Holen, Einfügen, Editieren und Löschen von Benutzerdaten in der Datenbank an, übergibt dann die Ergebnisse zu der Komponente „badGUI“. (siehe Abschnitt 3.2.4.2) Wegen der Funktionalität von Chart (siehe Abschnitt 3.2.4.7) muss sie noch mit der Komponente „oma“ (siehe Abschnitt 3.2.4.4) zusammenarbeiten, um alle Daten, die ein Chart benötigt, aus der Datenbank zu holen.

#### 2.2.4.1.1 Spezifikation von bad

Die Verzeichnisstruktur von dieser Komponente ist folgendmassen definiert.

```
bad
|--- conf
|--- docs
|--- src
|--- templates
```

Unter dem Verzeichnis conf befinden sich die Konfigurationsdateien, die sind:

**bad.properties**: die die Eigenschaften der Komponente definiert.

**componentConfiguration.xml**: die die Konfiguration der Komponenten festlegt.

**roleConfiguration.xml**: die die Rollen von Avalon Service bestimmt.

Unter dem Verzeichnis docs befinden sich die JavaDocs der Source Code.

Da die Komponente bad nur Geschäftslogik enthält, hat sie gar nichts für die Anzeige, ist das Verzeichnis templates deswegen bei ihr auch leer.

Wichtig ist die Verzeichnisstruktur von src, unter diesem Verzeichnis befinden sich alle Source Code von dieser Komponente.

```
src
|--- java
|--- com
```

```

|--- gogol
    |--- bad
        |--- avalon
            |--- om

```

Die Verzeichnisstruktur hier ist auch nach der Konvention der vorgegebenen Entwicklungsumgebung definiert. Unter dem Unterverzeichnis avalon sind die folgenden Klassen definiert:

**Bad**: Schnittstelle, die den Rollenname definiert.

**BadComponent**: Schnittstelle, die die Avalon-Komponente erweitert, und alle Methoden, die zum Holen, Einfügen, Editieren und Löschen von Dateneinträgen in der Datenbank definiert.

**TorqueBadComponent**: Implementiert die Schnittstelle BadComponent.

Unter dem Verzeichnis om befinden sich die automatisch von Torque generierten Peer-Klassen, mit deren Hilfe der Zugriff zu der Datenbank ermöglicht ist.

Da es bei dem Zugriff zu der Datenbank Ausnahmen (Exceptions) auftauchen könnten, muss man auch bei dem Entwurf schon aufpassen, dass man eine eigene Exception-Class definiert, um die eventuell geworfenen Ausnahmen fangen zu können.

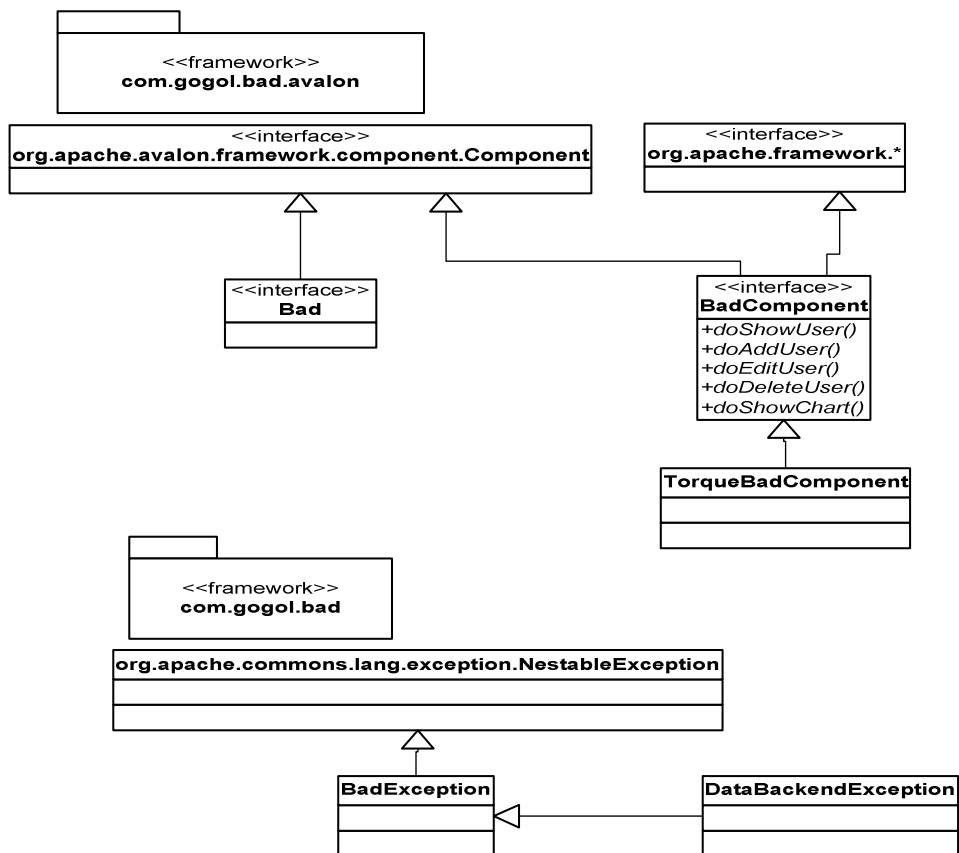


Abbildung 2-7 Struktur der Komponente bad

Das in der Abbildung 3-7 gezeichnete Diagramm entspricht der Klassen-Struktur der Komponente bad.

## 2.2.4.2 Basic Data GUI → badGUI

Diese Komponente enthält eine Action-Class, die mit den Benutzeranforderungen reagiert. Sie sammelt die Informationen von dem Benutzer, greift dann die Datenbank mit Hilfe von bad (siehe Abschnitt 3.2.4.1) zu, schickt danach die Ergebnisse zu der Web-Browser.

### 2.2.4.2.1 Spezifikation von badGUI

Die Verzeichnisstruktur von dieser Komponente ist folgendmassen definiert.

```
badGUI
|--- conf
|--- docs
|--- src
|--- templates
```

Unter dem Verzeichnis conf befinden sich die Konfigurationsdateien, die sind:

**badGUI.properties**: die die Eigenschaften der Komponente definiert. Wobei ein Pull-Tool Service (siehe Abschnitt 2.3.3) auch in dieser Datei definiert wird.

Unter dem Verzeichnis docs befinden sich die JavaDocs der Source Code.

Unter dem Verzeichnis templates befinden sich die Templates-Dateien zur Navigation und zum Anzeigen der Ergebnisse.

Wichtig ist die Verzeichnisstruktur von src, unter diesem Verzeichnis befinden sich alle Source Code von dieser Komponente.

```
src
|--- java
|--- com
|--- gogol
|--- badgui
|--- turbine
|--- avalon
|--- tools
|--- modules
|--- action
|--- secure
```

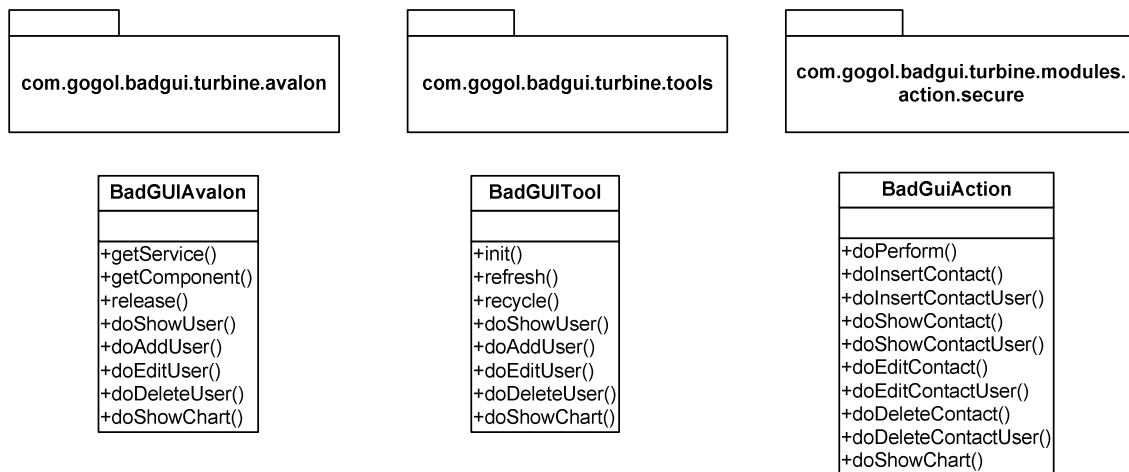
Unter dem Unterverzeichnis avalon befindet sich eine Klasse:

**BadGUIAvalon**: sie ist eine Fassade-Klasse<sup>1</sup> für diese Komponente  
 Unter dem Unterverzeichnis tools befindet sich das Pull-Tool:

**BadGUITool**: Diese Klasse bietet solche Methoden für die Templates an, damit die Templates die Ergebnisse nahher direkt anzeigen können.

Unter dem Unterverzeichnis secure befindet sich eine Action-Class:

**BadGUIAction**: Hier ist die Stelle, wo die Integrität und Korrektheit der Daten aus den Formularen nachgeprüft werden. Wenn irgendeine Data fehlt bzw. nicht korrekt ist, dann wird der Benutzer zu dem Formular zurückgeleitet, sobald er den Fehler behoben hat, werden die Daten dann in der Datenbank abgespeichert.



**Abbildung 2-8 Struktur der Komponente badGUI**

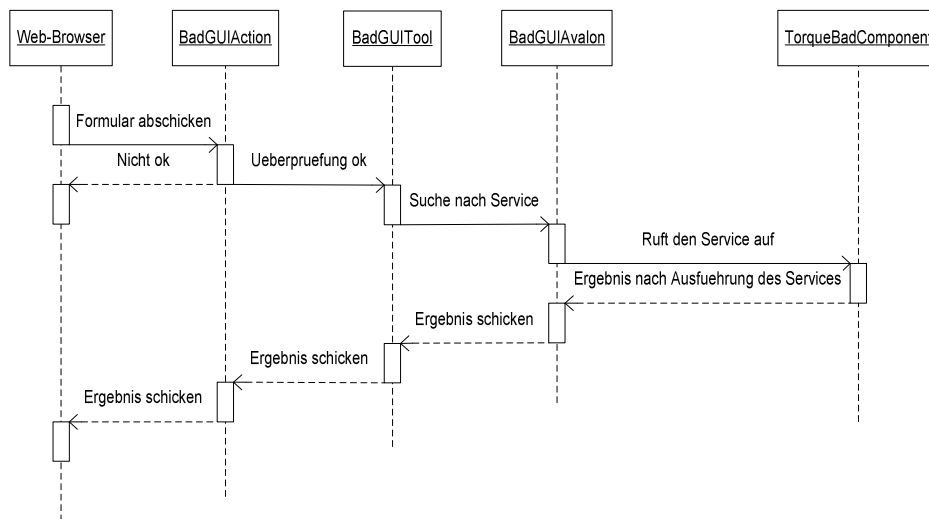
Das in der Abbildung 3-8 gezeichnete Diagramm entspricht der Klassen-Struktur der Komponente badGUI.

### 2.2.4.3 Zusammenhang zwischen bad und badGUI

Das in der Abbildung 3-9 gezeichnete Diagramm erklärt die Beziehung zwischen den Komponenten bad und badGUI.

Nachdem der Benutzer das Formular unter der Web-Browser aufgefüllt hat, und auf die Taste „Abschicken“ drückt, wird die Action-Class aufgerufen. Die Action-Class überprüft vor allem die Integrität und die Korrektheit der ausgefüllten Daten. Wenn es irgendeiner Fehler auftaucht, wird dann dem Benutzer das Formular wieder angezeigt, und zwar mit Fehlerquellen. Nach der Korrektur der Daten oder wenn die Daten schon korrekt sind, dann wird die Pull-Tool aufgerufen, die Pull-Tool sucht dann den entsprechenden Avalon Service, sobald der Service gefunden wird, wird er gleich ausgeführt. (Die Services werden von der Komponente bad angeboten) Egal ob der Service erfolgreich oder erfolglos ausgeführt ist, wird ein Ergebnis zurückgeliefert, und der Benutzer sieht es gleich in der Web-Browser.

<sup>1</sup> Eine Fassade-Klasse ist eine aufrufende Klasse, die die Services von bad aufruft..



**Abbildung 2-9 Anwendungssicht bad und badgui**

## 2.2.4.4 Order Management → oma

„oma“ ist auch eine Komponente, die den Zugriff zu der Datenbank hat. Sie bietet die Methoden zum Holen, Einfügen, Editieren und Löschen von Auftragsdaten in der Datenbank an, übergibt dann die Ergebnisse zu der Komponente „omaGUI“ (siehe Abschnitt 3.4.4). Wegen der Funktionalität von Report (siehe Abschnitt 3.4.6) muss sie noch mit der Komponente „bad“ (siehe Abschnitt 3.4.1) zusammenarbeiten, um alle Daten, die ein Bericht benötigt, aus der Datenbank zu holen.

### 2.2.4.4.1 Spezifikation von oma

Die Verzeichnisstruktur von dieser Komponente ist folgendmassen definiert.

```

oma
  |-- conf
  |-- docs
  |-- src
  |-- templates
  
```

Unter dem Verzeichnis conf befinden sich die Konfigurationsdateien, die sind:

**oma.properties**: die die Eigenschaften der Komponente definiert.  
**componentConfiguration.xml**: die die Konfiguration der Komponenten festlegt.  
**roleConfiguration.xml**: die die Rollen von Avalon Service bestimmt.

Unter dem Verzeichnis docs befinden sich die JavaDocs der Source Code.

Da die Komponente *oma* nur Geschäftslogik enthält, hat sie gar nichts für die Anzeige, ist das Verzeichnis *templates* deswegen bei ihr auch leer.

Wichtig ist die Verzeichnisstruktur von *src*, unter diesem Verzeichnis befinden sich alle Source Code von dieser Komponente.

```
src
|-- java
  |-- com
    |-- gogol
      |-- oma
        |-- avalon
          |-- om
```

Die Verzeichnisstruktur hier ist auch nach der Konvention der vorgegebenen Entwicklungsumgebung definiert. Unter dem Unterverzeichnis *avalon* sind die folgenden Klassen definiert:

***Oma***: Schnittstelle, die den Rollenname definiert.

***OmaComponent***: Schnittstelle, die die Avalon-Komponente erweitert, und alle Methoden, die zum Holen, Einfügen, Editieren und Löschen von Dateneinträgen in der Datenbank definiert.

***TorqueOmaComponent***: Implementiert die Schnittstelle *OmaComponent*.

Unter dem Verzeichnis *om* befinden sich die automatisch von Torque generierten Peer-Klassen, mit deren Hilfe der Zugriff zu der Datenbank ermöglicht ist.

Da es bei dem Zugriff zu der Datenbank Ausnahmen (Exceptions) auftauchen könnten, muss man auch bei dem Entwurf schon aufpassen, dass man eine eigene Exception-Class definiert, um die eventuell geworfenen Ausnahmen fangen zu können.

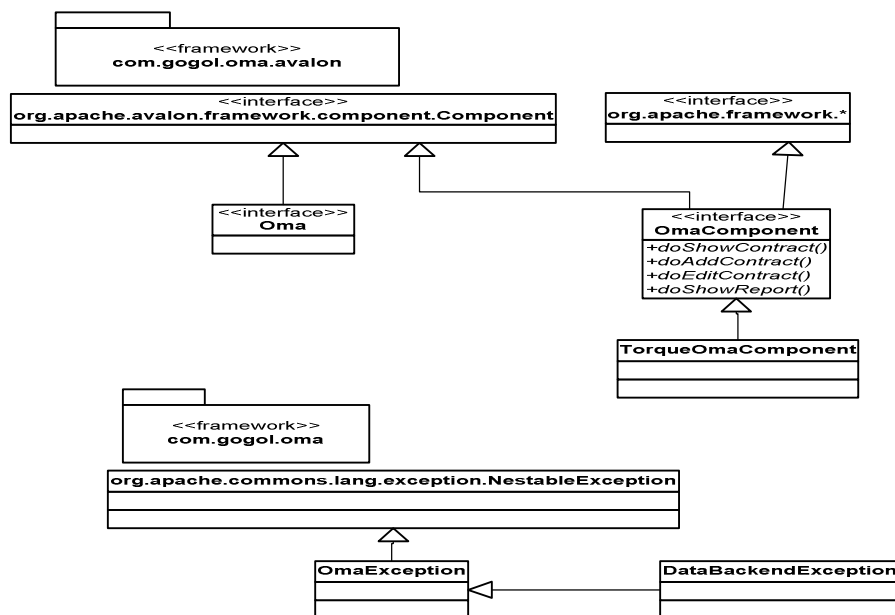


Abbildung 2-10 Struktur der Komponente oma

Das in der Abbildung 3-10 gezeichnete Diagramm entspricht der Klassen-Struktur der Komponente oma.

## 2.2.4.5 Order Management GUI → omaGUI

Diese Komponente enthält eine Action-Class, die mit den Benutzeranforderungen reagiert. Sie sammelt die Informationen von dem Benutzer, greift dann die Datenbank mit Hilfe von oma (siehe Abschnitt 3.4.3) zu, schickt danach die Ergebnisse zu der Web-Browser.

### 2.2.4.5.1 Spezifikation von omaGUI

Die Verzeichnisstruktur von dieser Komponente ist folgendmassen definiert.

```
omaGUI
|--- conf
|--- docs
|--- src
|--- templates
```

Unter dem Verzeichnis conf befinden sich die Konfigurationsdateien, die sind:

**omaGUI.properties**: die die Eigenschaften der Komponente definiert. Wobei ein Pull-Tool Service (siehe Abschnitt 3.1.5) auch in dieser Datei definiert wird.

Unter dem Verzeichnis docs befinden sich die JavaDocs der Source Code.

Unter dem Verzeichnis templates befinden sich die Templates-Dateien zur Navigation und zum Anzeigen der Ergebnisse.

Wichtig ist die Verzeichnisstruktur von src, unter diesem Verzeichnis befinden sich alle Source Code von dieser Komponente.

```
src
|--- java
|--- com
|--- gogol
|--- omagui
|--- turbine
|--- avalon
|--- tools
|--- modules
|--- action
|--- secure
```

Unter dem Unterverzeichnis avalon befindet sich eine Klasse:

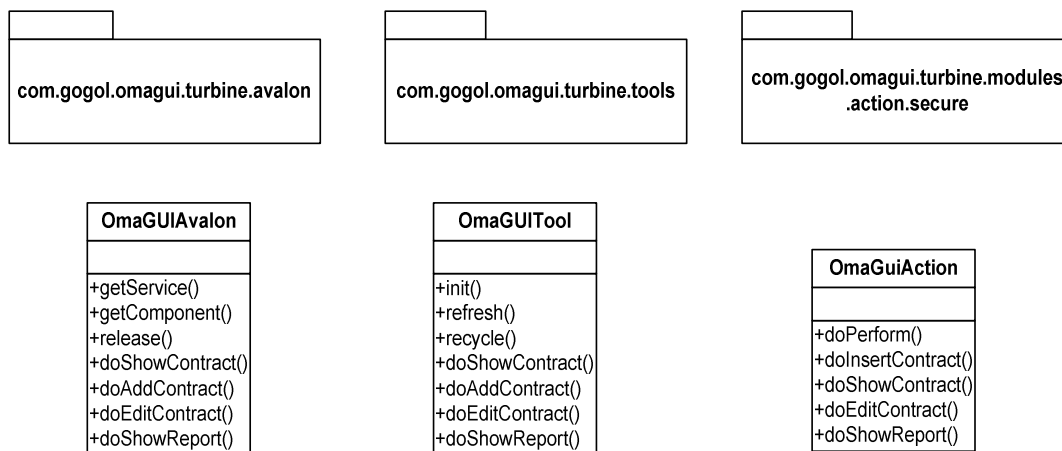
**OmaGUIAvalon**: sie ist eine Fassade-Klasse<sup>1</sup> für diese Komponente

Unter dem Unterverzeichnis tools befindet sich das Pull-Tool:

**OmaGUITool**: Diese Klasse bietet solche Methoden für die Templates an, damit die Templates die Ergebnisse nachher direkt anzeigen können.

Unter dem Unterverzeichnis secure befindet sich eine Action-Class:

**OmaGUIAction**: Hier ist die Stelle, wo die Integrität und Korrektheit von den Daten aus den Formularen nachgeprüft werden. Wenn irgendeine Data fehlt bzw. nicht korrekt ist, dann wird der Benutzer zu dem Formular zurückgeleitet, sobald er den Fehler behoben hat, werden die Daten dann in der Datenbank abgespeichert.



**Abbildung 2-11 Struktur der Komponente omaGUI**

Das in der Abbildung 3-11 gezeichnete Diagramm entspricht der Klassen-Struktur der Komponente *omaGUI*.

## 2.2.4.6 Zusammenhang zwischen oma und omaGUI

Das in der Abbildung 3-12 gezeichnete Diagramm erklärt die Beziehung zwischen den Komponenten oma und omaGUI.

Nachdem der Benutzer das Formular unter der Web-Browser aufgefüllt hat, und auf die Taste „Abschicken“ drückt, wird die Action-Class aufgerufen. Die Action-Class überprüft vor allem die Integrität und die Korrektheit der ausgefüllten Daten, wenn es irgendeiner Fehler auftaucht, wird dann dem Benutzer das Formular wieder angezeigt, und zwar mit Fehlerquellen. Nach der Korrektur der Daten oder wenn die Daten schon korrekt sind, dann wird die Pull-Tool aufgerufen, die Pull-Tool sucht dann den entsprechenden Avalon Service, sobald der Service gefunden wird, wird er gleich ausgeführt. (Die Services werden von der Komponente oma angeboten) Egal

<sup>1</sup> Eine Fassade-Klasse ist eine aufrufende Klasse, die die Services von der Komponente oma aufruft.

ob der Service erfolgreich oder erfolglos ausgeführt ist, wird ein Ergebnis zurückgeliefert, und der Benutzer sieht es gleich in der Web-Browser.

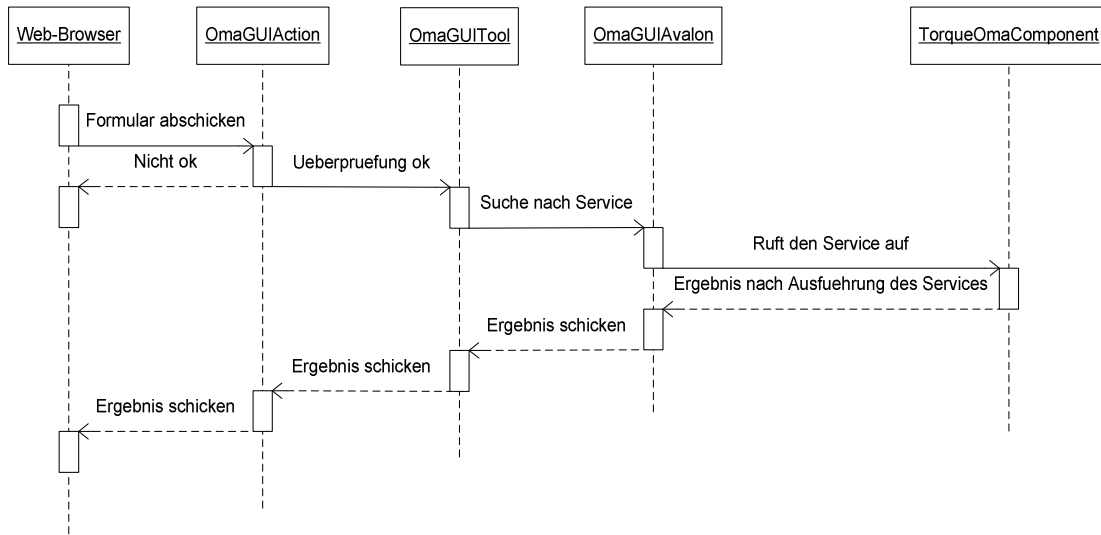
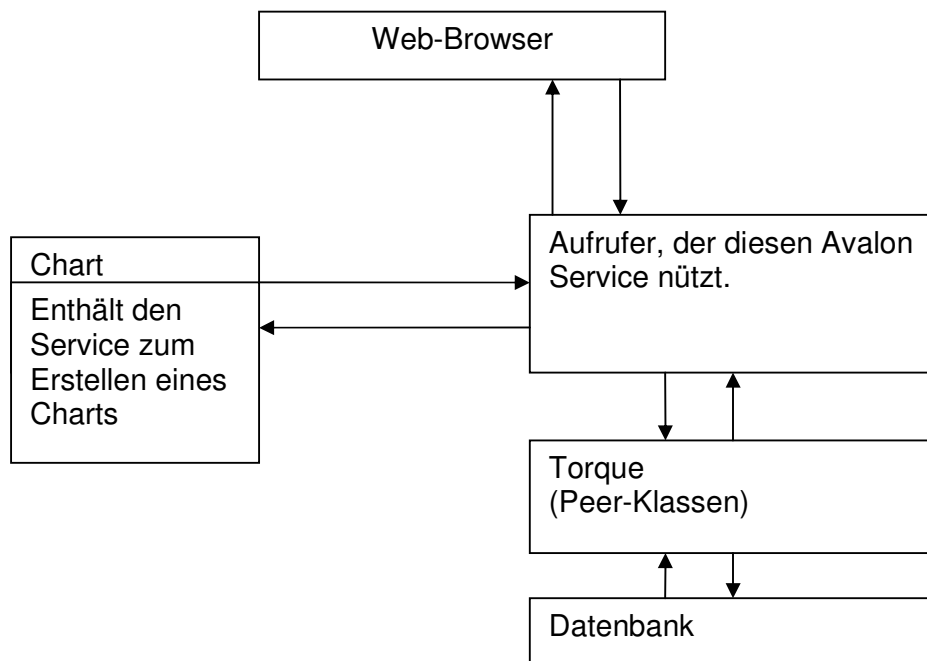


Abbildung 2-12 Anwendungssicht oma und omagui

### 2.2.4.7 Chart

Für das Erstellen von Charts ist eine Komponente Chart entworfen, die nur Business Logik enthält, und keine Templates hat. Also sie bekommt als Eingabe eine Datenmenge(Daten über Kategorien und numerische Daten, siehe unten), dann wird einen Chart in JPEG erstellt, auf der Festplatte gespeichert, und dann wird das URL dieser Datei zu dem Aufrufer<sup>1</sup> zurückgegeben.



<sup>1</sup> Der Aufrufer hier sind die Komponenten bad und badGUI.

## Abbildung 2-13 Grobe Struktur von Komponente Chart

Die Komponente *Chart* implementiert einen Avalon-Service für das Erstellen von Charts. Sie enthält nur die Geschäftslogik, und arbeitet passiv, d.h. sie arbeitet nur, wenn sie von einem Aufrufer aufgerufen wird. Der Aufrufer ist hier die Komponente *badGUI* gemeint, da *badGUI* der Controller-Komponente nach dem MVC-Paradigma (siehe Abschnitt 3.1.3) entspricht, ruft sie zunächst die Model-Komponente (*bad*) auf, um die benötigten Daten aus der Datenbank zu holen, dann werden die Daten gleich nach der Komponente *Chart* transportiert, um dort die Charts zu erstellen. Der generierte Chart wird dann mit Hilfe von *badGUI* sofort in der Web-Browser angezeigt.

Wichtig bei dieser Komponente ist, dass die Datenmenge vorher geordnet werden muss, (siehe Abschnitt 3.2.4.7.1), damit etwas sinnvolles im Chart angezeigt wird.

Es sind zwei Varianten von Charts, nämlich Balkenchart und Tortenchart, die eigentlich über dieselbe Datenmenge arbeiten, bloß verschiedene Darstellungsart haben.

### 2.2.4.7.1 Spezifikation von Chart

Die Verzeichnisstruktur von dieser Komponente ist folgendmassen definiert.

```
Chart
|---conf
|---docs
|---src
|---templates
```

Unter dem Verzeichnis *conf* befinden sich die Konfigurationsdateien, die sind:

**chart.properties**: die die Eigenschaften der Komponente definiert.

**componentConfiguration.xml**: die die Konfiguration der Komponenten festlegt.

**roleConfiguration.xml**: die die Rollen von Avalon Service bestimmt.

Unter dem Verzeichnis *docs* befinden sich die JavaDocs der Source Code.

Da die Komponente Chart nur Geschäftslogik enthält, hat sie gar nichts für die Anzeige, ist das Verzeichnis *templates* deswegen bei ihr auch leer.

Wichtig ist die Verzeichnisstruktur von *src*, unter diesem Verzeichnis befinden sich alle Source Code von dieser Komponente.

```
src
|-- java
|   |-- com
|       |-- gogol
|           |-- chart
|               |-- avalon
```

Da es bei der Generierung von Charts Ausnahmen (Exceptions) auftauchen könnten, muss man auch bei dem Entwurf schon aufpassen, dass man eine eigene Exception-Class definiert, um die eventuell geworfenen Ausnahmen fangen zu können.

Unter dem Unterverzeichnis avalon sind folgende Klassen definiert:

**Chart** eine Schnittstelle, die den Rollennamen definiert.

**ChartComponent** eine Schnittstelle, die die Avalon-Komponente erweitert, und folgende Methoden definiert:

- **File createBarChart(String[] term, String[] kategorie, double[] daten, File tempFile)**: Eine Kategorie kann eventuell mehrere Terme enthalten, z.B. die Kategorien bezeichnen die Monate, und die Terme bezeichnen verschiedene Umsätze, wie Umsatz von Büchern oder Umsatz von Zeitungen. In diesem Fall werden zwei Balken bei jeder Kategorie angezeigt, und alle Kategorien werden ebenfalls dargestellt, es sind also insgesamt 24 Balken im Chart. Aber eine Kategorie kann auch nur einen einzigen Term enthalten. In diesem Fall sind es nur 12 Balken im Chart. Die Daten enthalten alle Daten von allen Balken. Die Methode erzeugt einen Balkenchart, speichert ihn in die angegebene Datei (tempFile), und gibt die Datei als Ausgabe zurück.
- **File createPieChart(String[] term, String[] kategorie, double[] daten, File tempFile)**: Die Struktur bei Tortenchart ist anders, unter Kategorien unterteilt sie keine weiteren Terme, sie setzt voraus, dass eine Kategorie nur einen Term besitzt. Wenn es aber mehrere Terme unter einer Kategorie stehen, dann werden alle Daten von allen diesen Termen zu dieser Kategorie zusammengezählt, danach wird die Methode **createPieChart(String[] kategorie, double[] daten, File tempFile)** aufgerufen. Diese Methode erzeugt einen Tortenchart, speichert ihn in die angegebene Datei (tempFile), und gibt die Datei als Ausgabe zurück.
- **File createPieChart(String[] kategorie, double[] daten, File tempFile)**: erzeugt mit den gegebenen Kategorien und Daten einen Tortenchart, speichert ihn in die angegebene Datei (tempFile), und gibt die Datei als Ausgabe zurück.

**ChartComponentImp** implementiert die Schnittstelle **ChartComponent** von oben.

Das in der Abbildung 3-14 gezeichnete Diagramm entspricht der Klassen-Struktur der Komponente Chart.

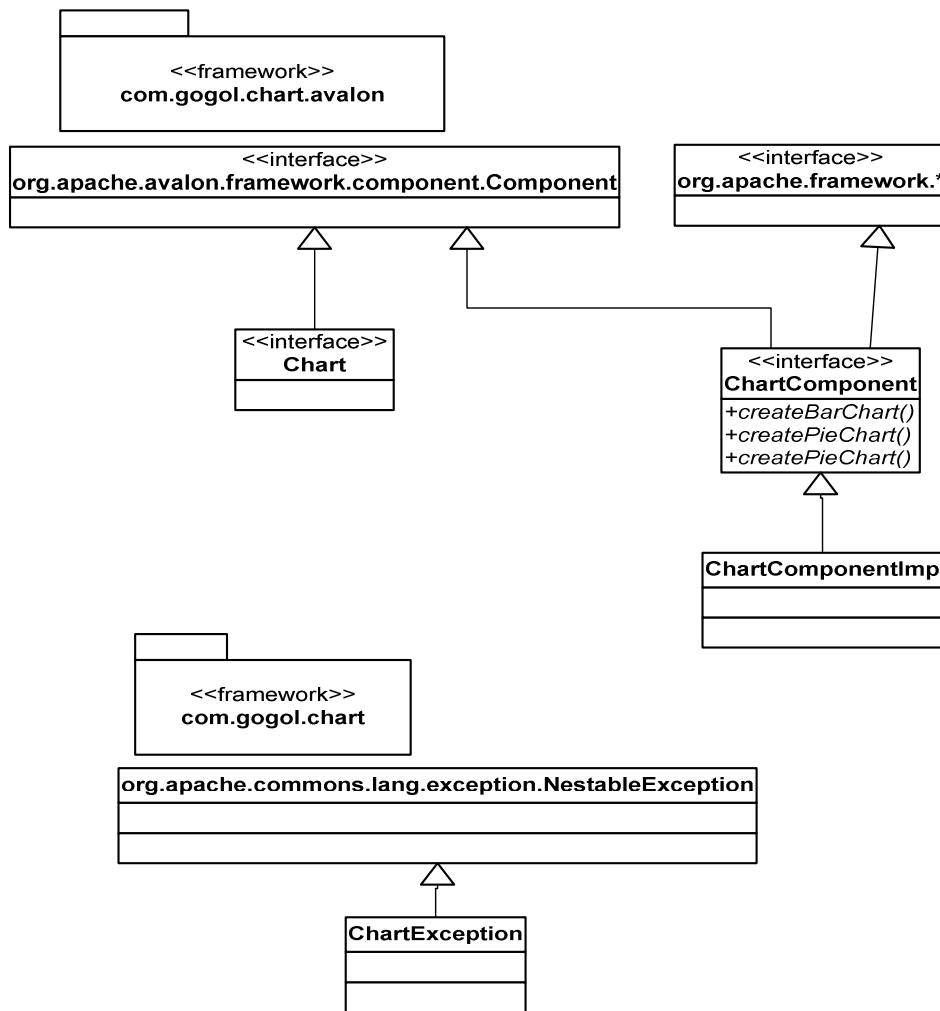


Abbildung 2-14 Klassen-Struktur von Komponente Chart.

### 2.2.4.8 Report

Für das Erstellen von Berichten ist eine Komponente Report entworfen, die reine Geschäftslogik enthält, und keine Templates hat. Also sie bekommt als Eingabe eine Datenmenge (Daten über alle Einträge von einer Tabelle, siehe unten), dann wird ein Bericht in HTML erstellt, auf der Festplatte gespeichert, und die generierte Datei wird zurückgegeben. Sie bietet zusätzlich noch die Möglichkeit, dass man den Bericht in PDF speichern kann.

Die Komponente Report implementiert einen Avalon-Service für das Erstellen von Berichten. Sie enthält nur die Geschäftslogik, und arbeitet ebenfalls passiv, d.h. sie arbeitet nur, wenn sie von einem Aufrufer aufgerufen wird. Der Aufrufer ist hier die Komponente *omaGUI* gemeint, da *omaGUI* der Controller-Komponente nach dem MVC-Paradigma (siehe Abschnitt 3.1.3) entspricht, ruft sie zunächst die Model-Komponente (*oma*) auf, um die benötigten Daten aus der Datenbank zu holen, dann werden die Daten gleich nach der Komponente Report transportiert, um dort die

Berichte zu erstellen. Der generierte Bericht wird dann mit Hilfe von omaGUI sofort in der Web-Browser angezeigt.

Wichtig bei dieser Komponente ist, dass die Datenmenge zunächst in TableModel<sup>1</sup> umgewandelt werden muss, da das verwendete Open-Source Werkzeug JFreeReport<sup>2</sup> nur mit TableModel arbeitet. JFreeReport benötigt noch eine Datei für die Steuerung der Anzeige. Sie ist eine XML Datei, die die Struktur und das Aussehen von einem Bericht definiert. Mit den beiden Sachen kann man dann einen Bericht erstellen, obwohl er noch nicht zu sehen ist. JFreeReport kann dann seine Berichte in HTML und PDF speichern, und die HTML Datei wird dann lokal auf der Festplatte gespeichert, und die generierte Datei wird zurückgegeben. Der Benutzer kann noch zusätzlich fordern, dass er den Bericht in PDF haben will, dann erstellt die Komponente Report ebenfalls eine PDF Datei, analog wie bei der HTML Datei.

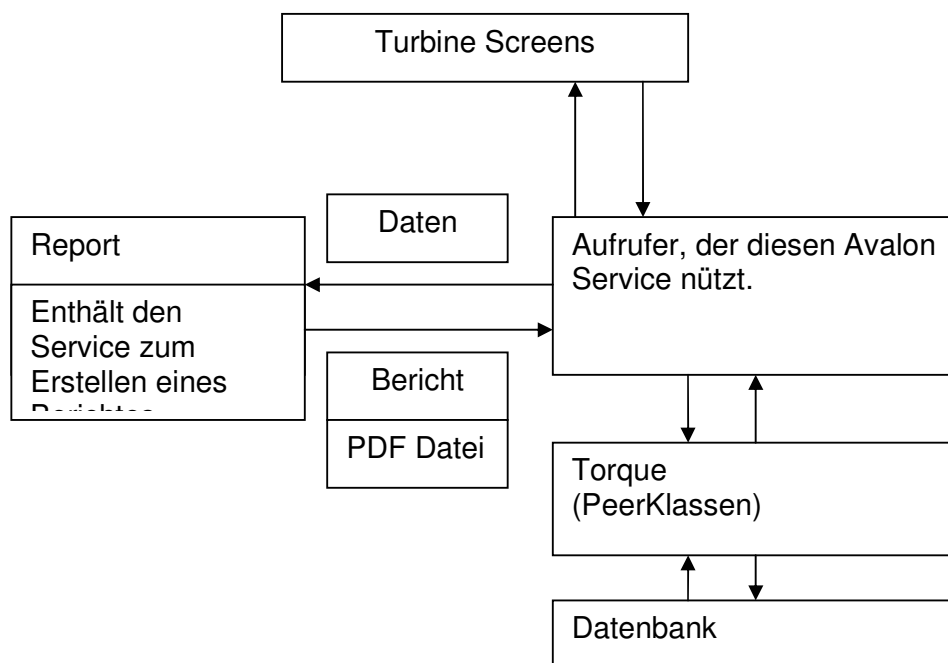


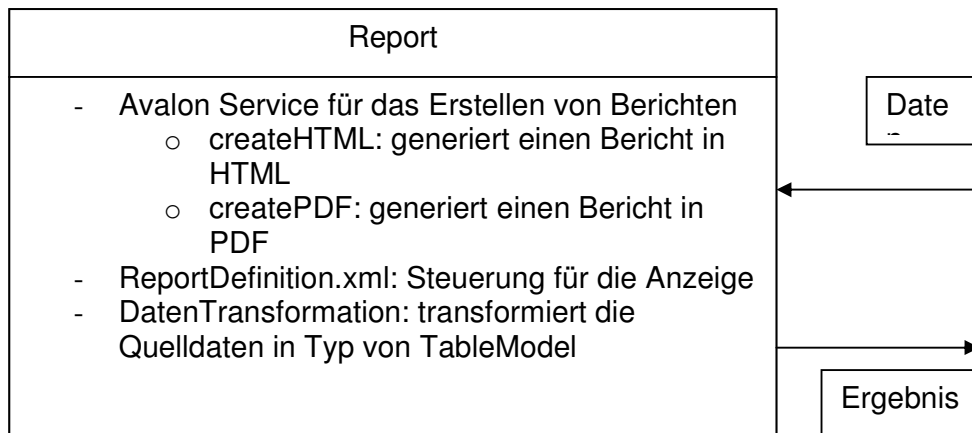
Abbildung 2-15 Grobe Struktur von Report

### 2.2.4.8.1 Spezifikation von Report

Das in der Abbildung 3-16 gezeichnete Diagramm erklärt die verfeinerte Struktur von Report. Wichtig dabei ist, dass die Quelldaten zunächst in TableModel umgewandelt werden muss. Um die Arbeit von der aufrufenden Komponente zu sparen, wird dieser Vorgang innerhalb von Report stattgefunden.

<sup>1</sup> TableModel basiert auf JTable, die aus dem Packet javax.swing.JTable stammt.

<sup>2</sup> Siehe [www.jfree.org](http://www.jfree.org)



**Abbildung 2-16 Verfeinerte Struktur von Report**

Die Verzeichnisstruktur von dieser Komponente ist folgendmassen definiert.

```

Report
|---conf
|---docs
|---src
|---templates
  
```

Unter dem Verzeichnis conf befinden sich die Konfigurationsdateien, die sind:

- report.properties**: die die Eigenschaften der Komponente definiert.
- componentConfiguration.xml**: die die Konfiguration der Komponenten festlegt.
- roleConfiguration.xml**: die die Rollen von Avalon Service bestimmt.

Unter dem Verzeichnis docs befinden sich die JavaDocs der Source Code.

Da die Komponente Report nur Geschäftslogik enthält, hat sie gar nichts für die Anzeige, ist das Verzeichnis templates deswegen bei ihr auch leer.

Wichtig ist die Verzeichnisstruktur von src, unter diesem Verzeichnis befinden sich alle Source Code von dieser Komponente.

```

src
|
|-- java
|   |-- com
|       |-- gogol
|           |-- report
|               |-- avalon
  
```

Da es bei der Generierung von Berichten Ausnahmen (Exceptions) auftauchen könnten, muss man auch bei dem Entwurf schon aufpassen, dass man eine eigene

Exception-Class definiert, um die eventuell geworfenen Ausnahmen fangen zu können.

Unter dem Unterverzeichnis avalon sind folgende Klassen definiert:

**Report:** eine Schnittstelle, die den Rollennamen definiert.

**DataTransform:** eine Klasse, die die Klasse AbstractTableModel erweitert. Die Hauptaufgabe dieser Klasse ist, sie bekommt im Konstruktor die Datenmenge, und speichert die um, zwar in Typ von TableModel, damit der Vorgang zum Erstellen von einem Bericht erleichtert wird.

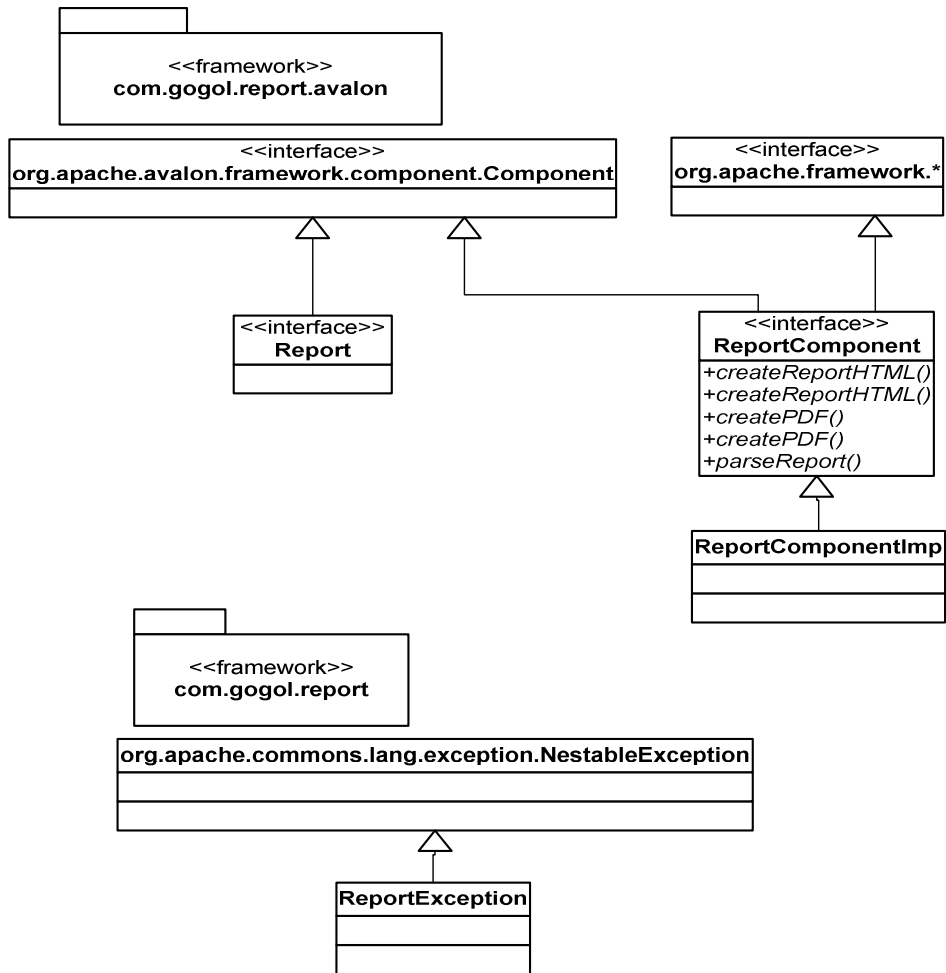
Die Datenmenge ist so abgespeichert: sie ist eine Struktur von Vektor über Vektoren, in dem ersten inneren Vektor werden die Spaltennamen von der Tabelle übergeben, dann werden nach der Anzahl von Spalten die entsprechenden Anzahl von String Arrays erzeugt, die die Daten von den weiteren inneren Vektoren halten. Ab dem zweiten inneren Vektor werden die echten Daten gespeichert, und alle Daten sind im Typ von String gespeichert. Danach wird durch eine Schleife die Daten aus den einzelnen inneren Vektoren herausgelesen, und zu den String Arrays zugeordnet. Die Klasse DataTransform implementiert noch die Methoden: *getColumnCount()*, *getColumnName(final int column)*, *getValueAt(final int row, final int column)* von AbstractTableModel.

**ReportDefinition.xml:** eine XML Datei, die die Struktur und das Aussehen von dem Bericht definiert.

**ReportComponent:** eine Schnittstelle, die Avalon-Komponente erweitert, und folgende Methoden definiert:

- **File createReportHTML(Vector data, File tempFile):** diese Methode sucht zuerst die default XML Datei(es sind einige XML Dateien in der Komponente vorhanden, in diesem Fall wird die erste XML Datei benutzt, der Benutzer kann auch seine eigene XML Datei angeben), parst die Datei durch *parseReport()* (s. unten), erzeugt dann eine Instanz von *DataTransform*, übergibt ihr den Vektor von Daten. Mit der verarbeiteten Daten und der XML Datei wird ein Bericht in HTML erzeugt, die HTML-Datei wird dann in die angegebene Datei (tempFile) gespeichert, und diese Datei wird zurückgegeben.
- **File createReportHTML(Vector data, String xmlURL, File tempFile):** eine Alternative Methode wie oben, aber die XML Datei wird von dem Aufrufer spezifiziert.
- **File createPDF(Vector data, File tempFile):** analog wie bei der HTML Version, wird hier aber eine PDF Datei erstellt. Die PDF-Datei wird dann in die angegebene Datei (tempFile) gespeichert, und diese Datei wird zurückgegeben.
- **File createPDF(Vector data, String xmlURL, File tempFile):** eine Alternative Methode wie oben, aber die XML Datei wird von dem Aufrufer spezifiziert.
- **JFreeReport parseReport(final URL templateURL):** Der Parser-Prozess von der XML Datei.

**ReportComponentImp**: Implementiert die Schnittstelle *ReportComponent* von oben.



**Abbildung 2-17** Klassen-Struktur von Komponente Report.

Das in der Abbildung 3-17 gezeichnete Diagramm entspricht der Klassen-Struktur der Komponente Report.

## 2.3 Implementierung

Bei der Implementierung wurde versucht, die Design-Vorgaben aus Kapitel 3.2 so streng wie möglich einzuhalten. Dies ist weitestgehend gelungen. Es waren lediglich kleinere Namensänderungen notwendig (z.B. um Mehrdeutigkeiten zu vermeiden). Des Weiteren wurden manchmal zusätzliche Methoden eingeführt, um den Zugriff auf die Schnittstellen zu vereinfachen oder ein mehrmaliges Auftreten von bestimmten Funktionalitäten in einer Methode zu bündeln.

### 2.3.1 Die Entwicklungsumgebung

Die Entwicklung hat folgende Ausstattung:

- Windows XP Professional als Betriebssystem
- PostgreSQL 7.2 als Mehrbenutzer-Datenbank
- Tomcat 4.29 als Servlet-Container und Web-Server
- Java Development Kit 1.4.2
- Ant 1.5
- PHP 4.3.3
- Apache Server als Web-Server für PHP
- Microsoft Access als Einbenutzer-Datenbank
- Turbine 2.3
- Torque 3.1
- Velocity 1.3.1
- Avalon-Framework 4.1.4
- JFreeChart 0.9.15
- JFreeReport 0.8.47

### 2.3.2 Verwaltungsbereich

In diesem Abschnitt werden die Verwaltung und die Konfiguration der Services genauer beschrieben. Zuerst wird die Struktur von einem Pull-Tool erklärt, dann wird die Konfiguration von Pull-Tool und den anderen Ressourcen erläutert.

#### Pull-Tool

Bei der Implementierung der Pull-Tools wurde der *Request-Scope* gewählt. Dadurch ist es möglich eventuelle Parameter, die mit einem Request geschickt werden (GET oder POST), direkt in dem Pull-Tool zu verarbeiten und entsprechend darauf zu reagieren.

Ein Pull-Tool muss generell die beiden Interfaces *ApplicationTool* und *Recycleable* implementieren. Diese Interfaces definieren Schnittstellen zur Initialisierung und zur Wiederverwendung eines Pull-Tool-Objektes. Wiederverwendung bedeutet in diesem

Zusammenhang, dass während des Startvorgangs der Anwendung eine bestimmte Anzahl an Objekten instanziiert und in einem Pool abgelegt werden. Wird nun ein Objekt dieses Typs benötigt, kann Turbine auf diesen Pool von Objekten zurückgreifen. Dadurch wird das zeitintensive Instanzieren eines neuen Objektes während der Laufzeit vermieden.

Initialisiert wird ein Pull-Tool über folgende Methode:

```
public void init(Object data){
    this.data = (RunData) data;
}
```

Bei dem gewählten Request-Scope wird dabei ein Objekt des Typs RunData übergeben, das die Laufzeitinformationen von Turbine enthält. Dieses wird zur späteren Verwendung in einer globalen Variable data zwischengespeichert.

Damit dieses Pull-Tool in den Templates verfügbar ist, muss ein Eintrag in die Datei `TurbineResources.template` vorgenommen werden. Z.B.

```
tool.request.omagui = com.gogol.omagui.turbine.tools.OmaGUITool
```

Die Syntax, die dabei verwendet wird, ist recht einfach. Der Eintrag fängt immer mit `tool` an, als Kennzeichen, dass es sich um ein Pull-Tool handelt. Darauf folgt der gewählte Scope und ein frei wählbarer, eindeutiger Name. Der Zugriff innerhalb eines Templates ist dann folgendermaßen möglich:

```
#set($comp = $omagui.getComponent())
Der Name der Komponente ist<br />
$comp.getName()
```

Wobei die Methode `getComponent()` eine Methode aus dem Pull-Tool stammt, der Rückgabetyt dieser Methode ist wieder ein Objekt. Das Objekt hat ein Attribut *Name*, dessen Wert man durch den Aufruf der Methode `getName()` bekommen kann.

## Neugenerierung der Konfigurationsdateien

Turbine wird über die Datei `TurbineResources.properties` konfiguriert. Diese wird durch den Ant-Task `update-tr-props` aus der Template-Datei `TurbineResources.template` erzeugt, wobei bestimmte Platzhalter durch in der Build-Datei `build.xml` definierte Werte ersetzt werden. D.h. Änderungen an der Konfiguration müssen immer in der Template-Datei vorgenommen werden. Anschließend ist dann ein manuelles Aufrufen von Ant notwendig. Dies erschwert die automatische Neugenerierung zur Laufzeit.

Aus diesem Grund werden alle zur Laufzeit zu ändernden Einstellungen in eine extra Datei z.B. `omagui.properties` ausgelagert. Diese wird durch die folgende Anweisung am Ende der ursprünglichen Datei angehängt:

```
include = omagui.properties
```

### 2.3.3 Ant Build-Datei

Die Build-Datei, die für die Entwicklung einer Komponente benötigt wird, ist auf Grund der sehr einfachen XML-Sprache leicht zu erstellen. In dieser XML-Sprache werden Variablen in der Form `${bezeichner}` verwendet. Diese Variablen werden in der Datei `build.properties` definiert. Die wichtigsten Einstellungen, die hier vorgenommen werden können, sind der Namen und die Version der Komponente, sowie der Ort der Web-Anwendung.

```
xaptor.component = omagui
xaptor.component.version = 1.0
xaptor.webapp.conf = ${xaptor.webapp}/WEB-INF/conf
```

Die Targets benutzen die von Ant standardmäßig zur Verfügung gestellten Tasks. Im Folgenden ist das Target `compile` aufgelistet:

```
<target name="compile" depends="makedirs" description="--> compiles the source
code">

  <javac srcdir="${src.java}" destdir="${build.classes}" debug="${debug}"
  deprecation="${deprecation}" optimize="${optimize}">

    <classpath refid="classpath"/>

  </javac>

</target>
```

Dabei wird über das Attribut `depends` des Elements `target` festgelegt, dass vor dessen Ausführung das Target `makedirs` aufgerufen wird. Zum Kompilieren der Java-Quellcode-Dateien wird der Task `javac` benutzt. Über dessen Attribute werden das Quell-Verzeichnis, das Ziel-Verzeichnis sowie verschiedene Compiler-Flags angegeben. Das Element `classpath` verweist über das Attribut `refid` auf den global definierten Klassenpfad.

Das Target `deploy` ist wie folgt implementiert:

```
<target name="deploy" description="--> deploys the component to the webapp"
depends="compile">

  <copy todir="${xaptor.webapp.templates}/${xaptor.component}">
  <fileset dir="${templates}" />
  </copy>
  <copy todir="${xaptor.webapp.conf}/${xaptor.component}">
  <fileset dir="${conf}" />
  </copy>
  <copy todir="${xaptor.webapp.classes}">
  <fileset dir="${build.classes}" />
  </copy>

</target>
```

Dabei wird wiederum über das Attribut *depends* definiert, dass vorher das Target *compile* ausgeführt werden soll. Dieses Target, das ja zum Kopieren der Dateien in die Anwendung dient, benutzt den sehr einfachen Task *copy*. Er kopiert alle Dateien und Verzeichnisse, die durch das Element *fileset* festgelegt werden, in das durch das Attribut *todir* angegebene Zielverzeichnis.

### 2.3.4 Oberfläche

Um einen etwas plastischeren Eindruck der Anwendung zu ermöglichen, ist in Abbildung 3-18 ein Screenshot der Anwendung abgebildet. Der Screenshot zeigt die Umsetzung des grundlegenden Oberflächenentwurfs, wobei der aktuelle Screen die Detailansicht der Komponente *bad* darstellt.

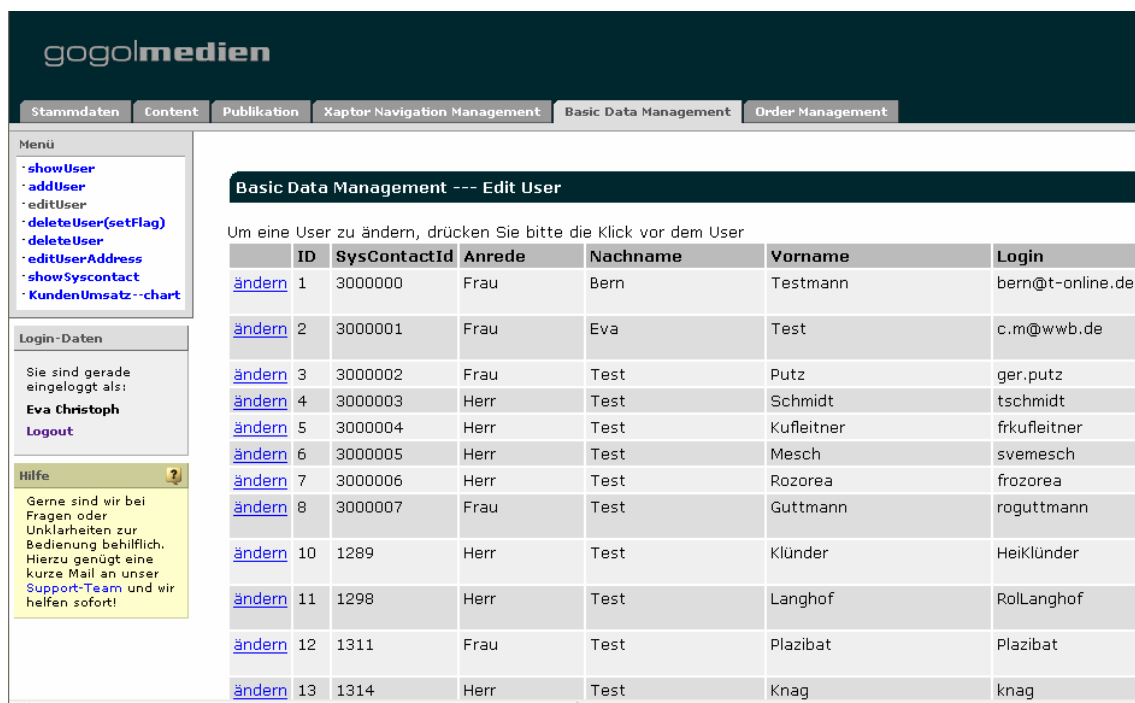


Abbildung 2-18 Ansicht der Komponente bad

### 2.3.5 Basic Data Management (bad und badGUI)

Die Hauptaufgabe dieser Komponente ist, die Geschäftslogik von den im Abschnitt 3.2.2 erwähnten Funktionalitäten zu realisieren. Wichtig dabei ist, dass die Geschäftslogik und die Anzeige getrennte Teile sind, d.h. die Komponente *bad* enthält nur die Geschäftslogik, und die andere Komponente *badGUI* ist für die Anzeige zuständig.

### 2.3.5.1 Anlegen von neuen Einträgen

*badGUI* bietet Formulare für den Benutzer, bei dem der Benutzer die Daten eingeben könnte. Für die Funktionalität von Anlegen eines neuen Benutzer muss man aufpassen, dass manche Datenfelder aus dem Formular Musseingabefelder sind<sup>1</sup>, und solcher Felder müssen ausgefüllt werden. Aus der Datenbankdefinition kann es sein, dass manche Felder mit numerischen Werten ausgefüllt werden müssen<sup>2</sup>. Die ausgefüllten Daten müssen gleich nach dem Abschicken in einer Methode der Action-Class nachgeprüft werden. Diese Methode enthält viele try-catch-Blocks<sup>3</sup>, falls es irgendein Fehler auftaucht, werden die Daten nicht weiter verarbeitet, sondern wird der Speichervorgang sofort abgebrochen, und der Benutzer wird zu dem vorherigen Formular zurückgeleitet.

Der folgende Code-Abschnitt zeigt den Fall, dass der Benutzer zu dem Formular mit den Fehlerquellen zurückgeleitet wird, wenn das Sonderfeld nicht mit numerischen Daten ausgefüllt wurde.

```
if(password == null || password.length()==0){
    redirect(data, context, "/screens/secure/XNMMissingRequiredInputs.vm");
    return;
}

if(passwordConfirm == null || passwordConfirm.length()==0){
    redirect(data, context, "/screens/secure/XNMMissingRequiredInputs.vm");
    return;
}

if(!(password.equals(passwordConfirm)))
{
    redirect(data, context, "/screens/secure/BADPasswordError.vm");
    return;
}

try {
    loginNumber = new Integer(login_number);
}catch (Exception e) {
    redirect(data, context, "/screens/secure/BADIntegerError.vm");
    return;
}

private void redirect(RunData data, Context context, String errorTemplate)
{
    context.put("errorTemplate", errorTemplate);
    data.setScreenTemplate("secure,BADAddUserForm.vm");
}
```

Die Methode *redirect()* hat die Aufgabe, dass die Anwendungskontext und die Daten

---

<sup>1</sup> Solche Felder sind als „not null“ in der Datenbank definiert.

<sup>2</sup> Solche Felder sind als „numeric“ in der Datenbank definiert.

<sup>3</sup> Je nach der Anzahl der Sonderfelder.

mit einer spezifizierten Template nach einer anderen Template übergeben werden können. Die Methode setScreenTemplate() legt fest, welche Template ist der Empfänger der Daten.

Um es leichter aus dem Formular sehen zu können, welche Felder Sonderfelder sind, muss man beim Erstellen eines Formulars aufpassen, dass die Sonderfelder mit Sondermarke gezeichnet werden müssen, und eine kurze Erklärung zu den Sondermarken muss es auch geben. (siehe Abbildung 2-19)

Der Fall könnte es auch auftauchen, dass der Benutzer versehentlich ein Feld mit falschem Wert ausgefüllt hat, dann auf Abschicken gedrückt, die Action-Class lässt den Fehler nicht zu, und der Benutzer wird zurückgeleitet, aber jetzt ist das Formular wieder leer, alle gerade ausgefüllten Daten muss man jetzt noch mal ausfüllen. Um das Formular so intelligent wie möglich zu machen, muss es noch in der Action-Class durch context zu Screen übergibt werden.

In Action-Class:

```
String salutation = data.getParameters().getString("salutation");
context.put("salutation", salutation);
```

```
String firstname = data.getParameters().getString("firstname");
context.put("firstname", firstname);
```

.....

In vm-Screen:

```
<tr>
  <th>Anrede$!hint</th>
  <td>
    #if($salutation)
      <input type="text" name="salutation" width="50" value="$salutation"/>
    #else
      <input type="text" name="salutation" width="50" />
    #end
  </td>
</tr>
```

```
<tr>
  <th>Vorname</th>
  <td>
    #if($firstname)
      <input type="text" name="firstname" width="50" value="$firstname"/>
    #else
      <input type="text" name="firstname" width="50" />
    #end
  </td>
</tr>
```

.....

Falls es bei der Überprüfung der Daten irgendein Fehler auftaucht, dann wird der Benutzer zu dem Formular zurückgeleitet, aber die Felder sind mit den gerade

ausgefüllten Werte ausgefüllt, die Fehlerquelle steht oberhalb von dem Formular, dann kann der Benutzer nach dem Hinweis den/die Fehler beheben.

**Abbildung 2-19** Formular zum Anlegen eines neuen Ansprechpartners

Deshalb muss man beim Erstellen eines Formulars mit Velocity aufpassen, dass das Attribut value nicht leer ist.

```
<tr>
  <th>Anrede $hint: </th>
  <td>
    <input type="text" name="salutation" value="$salutation">
  </td>
</tr>
```

Das Attribut value von text hat den Wert \$salutation, nach der Syntax von Velocity bedeutet es hier, wenn die Variable salutation nicht NULL ist, dann wird der Wert von salutation angezeigt, ansonsten wird es gar nichts angezeigt. Also bei der ersten Anzeige von dem Formular ist salutation noch nicht definiert, sie hat den Wert NULL, deshalb sieht der Benutzer, dass das Formular leer ist. Nachdem er das Formular ausgefüllt und abgeschickt hat, und es ein Fehler auftaucht, wird das selbe Formular noch mal angezeigt, allerdings bekommt die Variable salutation jetzt seinen Wert von der Anwendungskontext, die man vorher in der Action-Class mit übergeben hat. Deshalb sind die Felder aus einem Formular ab der zweiten Anzeige ausgefüllt.

Wenn die Überprüfung der im Formular ausgefüllten Daten erfolgreich abgeschlossen wird, dann wird die mit den Daten ausgefüllte Vektor weitergeleitet, und zwar zu der Komponente bad. bad bekommt als Eingabe die Daten, und greift die Datenbank mit den automatisch von Torque erstellten Peer-Klassen zu. Beim

Einfügen ist es relativ einfach, man muss bloß eine Instanz von den betreffenden Tabellen erzeugen, liest die Daten von der übergebenen Vektor aus, und speichert sie wieder in diese Instanz ab. Komplexere Fälle gibt es beim Einfügen auch, dass es mehrere Tabellen betreffen. In diesem Fall muss man für jede Tabelle eine Instanz erzeugen. Die Reihenfolge spielt hier eine wichtige Rolle, d.h. gibt es Fremdschlüssel bei der Tabellendefinition, dann müssen die Instanzen zuerst ausgefüllt und abgespeichert werden, die von anderen Tabellen referenziert werden.

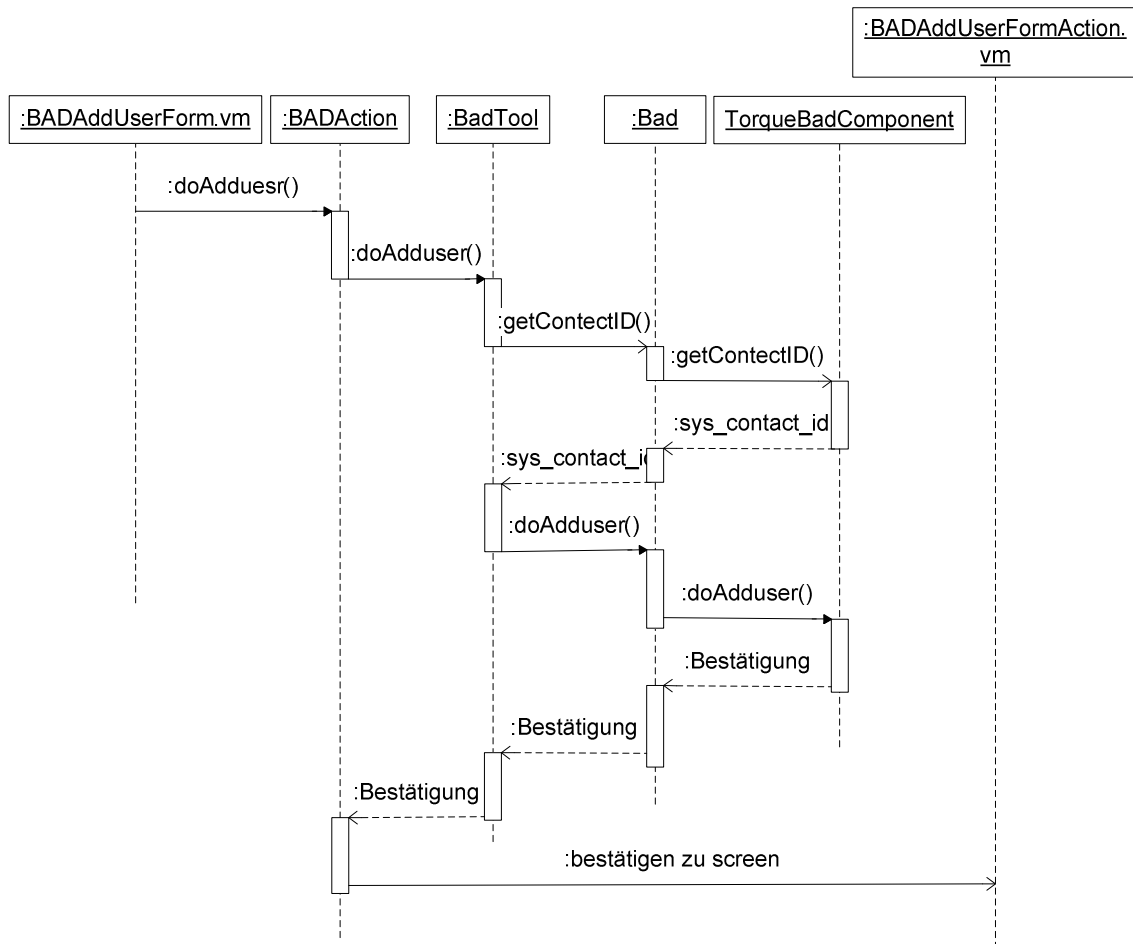


Abbildung 2-20 Ablauf addUser in der Komponente bad

### 2.3.5.2 Editieren von Einträgen

Das Editieren von den schon vorhandenen Einträgen ist ähnlich wie beim Anlegen eines neuen Eintrags. Allerdings muss man aufpassen, dass das System dem Benutzer die Möglichkeit anbietet, um einen gewünschten Eintrag aus einer Liste auszuwählen, und die detaillierten Informationen von diesem Eintrag werden dann in einem Formular angezeigt. Das Formular hier hat dieselbe Struktur wie beim Anlegen eines neuen Eintrags, allerdings sind die Felder mit den gespeicherten Werten von dem ausgewählten Eintrag ausgefüllt.

Nachdem Abschicken ist eine Überprüfung von den neu ausgefüllten Daten notwendig, da es möglich ist, dass der Benutzer durch das Editieren unzulässige Werte ausgefüllt hat. Im diesem Fall wird er auch zum Formular zurückgeleitet, um solche Werte zu ändern. Ansonsten werden die neuen Werte von diesem Eintrag gespeichert. Die Abbildung3-21 zeigt das Formular zum Editieren eines existierenden Eintrags.

### bad: eine User zu ändern

Mit folgendem Formular können Sie User Informationen ändern.

Kontaktsdaten neu anlegen

Die mit (\*) gekennzeichneten Felder sind Musseingaben.

|                             |   |
|-----------------------------|---|
| <b>UserId</b>               | <input type="text" value="2"/>              |
| <b>sysContactId (*)</b>     | <input type="text" value="3000001"/>        |
| <b>salutation (*)</b>       | <input type="text" value="Frau"/>           |
| <b>firstname</b>            | <input type="text" value="Eva"/>            |
| <b>lastname (*)</b>         | <input type="text" value="Test"/>           |
| <b>telephone</b>            | <input type="text" value="+49 99 889447"/>  |
| <b>mobile</b>               | <input type="text"/>                        |
| <b>fax</b>                  | <input type="text" value="+49 99 8196530"/> |
| <b>email</b>                | <input type="text" value="c.m@wwb.de"/>     |
| <b>internet</b>             | <input type="text"/>                        |
| <b>login (*)</b>            | <input type="text" value="c.m@wwb.de"/>     |
| <b>password (*)</b>         | <input type="text" value="3000001"/>        |
| <b>password confirm (*)</b> | <input type="text" value="3000001"/>        |
| <b>flag_active (*)</b>      | <input type="text" value="true"/>           |
| <b>flag_default (*)</b>     | <input type="text" value="true"/>           |
| <b>confirm_value</b>        | <input type="text"/>                        |
| <b>login_number</b>         | <input type="text" value="0"/>              |

**Abbildung 2-21** Formular zum Editieren eines existierenden Eintrags

### 2.3.5.3 Löschen von Einträgen

#### Logisches Löschen(set Active\_Flag):

Nach den vom Abschnitt 3.2 erwähnten Anforderungen sollen die Einträge durch das Löschen nur logisch gelöscht werden, d.h. die Einträge sind noch in der Datenbank vorhanden, aber nicht mehr in der Anwendung sichtbar. Dazu haben einige Tabellen in der Datenbank ein Flag<sup>1</sup>, und es hat am Anfang für alle Einträge den Wert „true“. Nach dem Löschen hat es den Wert „false“, und die Anzeige (siehe 3.3.5.4) zeigt nur alle Einträge mit dem Wert von „true“ an, deshalb sind die gelöschten Einträge nicht mehr in der Anwendung sichtbar.

<sup>1</sup> Eine Spalte in der Tabelle, dessen Typ als Boolean bezeichnet ist.

### Physikalisches löschen:

Zusätzlich wurde wirkliches Löschen auch Implementiert, um Einträge von Tabelle wirklich entfernen. Wenn man diese Funktion verwenden möchte, muss man vorsichtig feststellen, ob diese Einträge wirklich nicht mehr braucht ist. Denn diese Aktion ist nicht rücksetzbar.

### 2.3.5.4 Anzeigen von Einträgen

Die Anzeige von den existierenden Einträgen spielt in der Anwendung eine wichtige Rolle. Dadurch sind die Daten aus der Datenbank in der Anwendung geordnet und sichtbar. Sichtbar ist hier gemeint, dass die Daten zu sehen sind, aber es hier nicht möglich ist, die Daten zu ändern oder zu löschen. (siehe Abb. 3-22)

| Basic Data Management --- User Auflisten |              |        |          |            |                  |
|--|--------------|--------|----------|------------|------------------|
| ID                                       | SysContactId | Anrede | Nachname | Vorname    | Login            |
| 1  | 3000000      | Frau   | Bern     | Testmann   | bern@t-online.de |
| 2  | 3000001      | Frau   | Eva      | Test       | c.m@wwb.de       |
| 3  | 3000002      | Frau   | Test     | Putz       | ger.putz         |
| 4  | 3000003      | Herr   | Test     | Schmidt    | tschmidt         |
| 5  | 3000004      | Herr   | Test     | Kufleitner | frkufleitner     |
| 6  | 3000005      | Herr   | Test     | Mesch      | svemesch         |
| 7  | 3000006      | Herr   | Test     | Rozorea    | frozorea         |
| 8  | 3000007      | Frau   | Test     | Guttman    | roguttman        |
| 10                                       | 1289         | Herr   | Test     | Klunder    | HeiKlunder       |
| 11                                       | 1298         | Herr   | Test     | Langhof    | Rollanghof       |
| 12                                       | 1311         | Frau   | Test     | Plazibat   | Plazibat         |
| 13                                       | 1314         | Herr   | Test     | Knag       | knag             |

Abbildung 2-22 Anzeige von Einträgen

### 2.3.6 Order Management (oma und omaGUI)

Die Implementation von diesen Komponenten ist sehr ähnlich wie bei *bad* und *badGUI* (siehe Abschnitt 3.3.5), da sie von der Entwurf aus auch sehr ähnlich sind. Die Unterschiede hier sind:

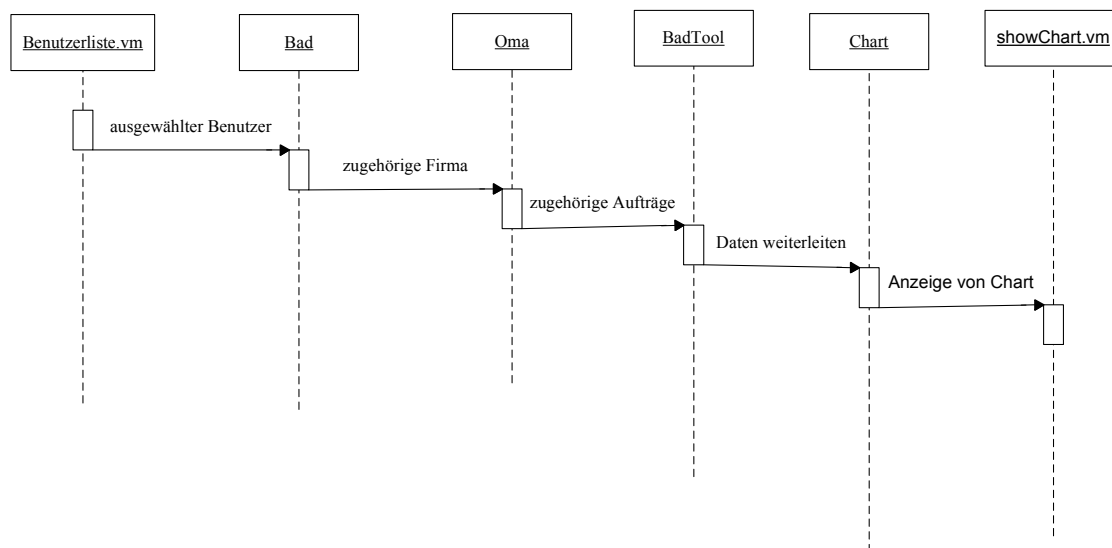
- Löschen von einem Auftrag ist nicht möglich, da es keine Flag-Definition<sup>1</sup> in den entsprechenden Tabellen gibt, wird es durch das Löschen ein Auftrag physikalisch gelöscht. Andererseits, die Aufträge werden in der alltäglichen Anwendung auch ganz selten gelöscht.
- Die betreffenden Tabellen sind andere Tabellen als bei *bad* und *badGUI*, deswegen haben die Formulare auch andere Inhalte.

<sup>1</sup> Eine Spalte in der Tabelle, dessen Typ als Boolean bezeichnet ist.

- Die Generierung von Berichten(siehe Abschnitt 3.2.4.8) findet hier statt, und die Generierung von Charts(siehe Abschnitt 3.2.4.7) findet bei bad und badGUI statt.

### 2.3.7 Generierung von Charts (chart)

Die Komponente Chart benötigt vor allem die Daten wegen der Generierung von Charts. Da die Daten dynamisch sind, müssen sie mit Hilfe von bad und oma aus der Datenbank gelesen werden.



**Abbildung 2-23 Ablauf von der Komponente Chart**

Das in der Abbildung 3-24 angezeigte Diagramm erklärt den Ablauf zum Sammeln der Daten. Also zuerst wird die Firma von dem ausgewählten Ansprechpartner mit Hilfe von bad bestimmt. Dann wird ebenfalls mit bad alle Ansprechpartner dieser Firma bestimmt. Wenn die Firma bekannt ist, dann kann man mit Hilfe von oma alle Aufträge für diese Firma auslesen. Die Aufträge sind dann nach Ansprechpartnern dieser Firma sortiert, die jeweiligen Umsätze werden summiert und jedem Ansprechpartner zugeordnet. Danach müssen die Daten noch nach der Definition von Chart (siehe Abschnitt 3.2.4.7) geordnet werden, damit Chart die Daten auslesen und weiterverarbeiten kann. Das Weiterverarbeiten ist so gemeint, da das Werkzeug JFreeChart<sup>1</sup> verschiedene Charts mit derselben Datenmenge darstellen kann, für unterschiedliche Charts muss man die Datenmenge aber so ändern, dass die Datenmenge die Anforderung von JFreeChart anpasst. Also die Hauptaufgabe von Chart ist, die Datenmenge je nach Diagramm anzupassen. Die Abbildung 3-24 zeigt ein Tortendiagramm.

<sup>1</sup> Siehe [www.jfree.org](http://www.jfree.org)

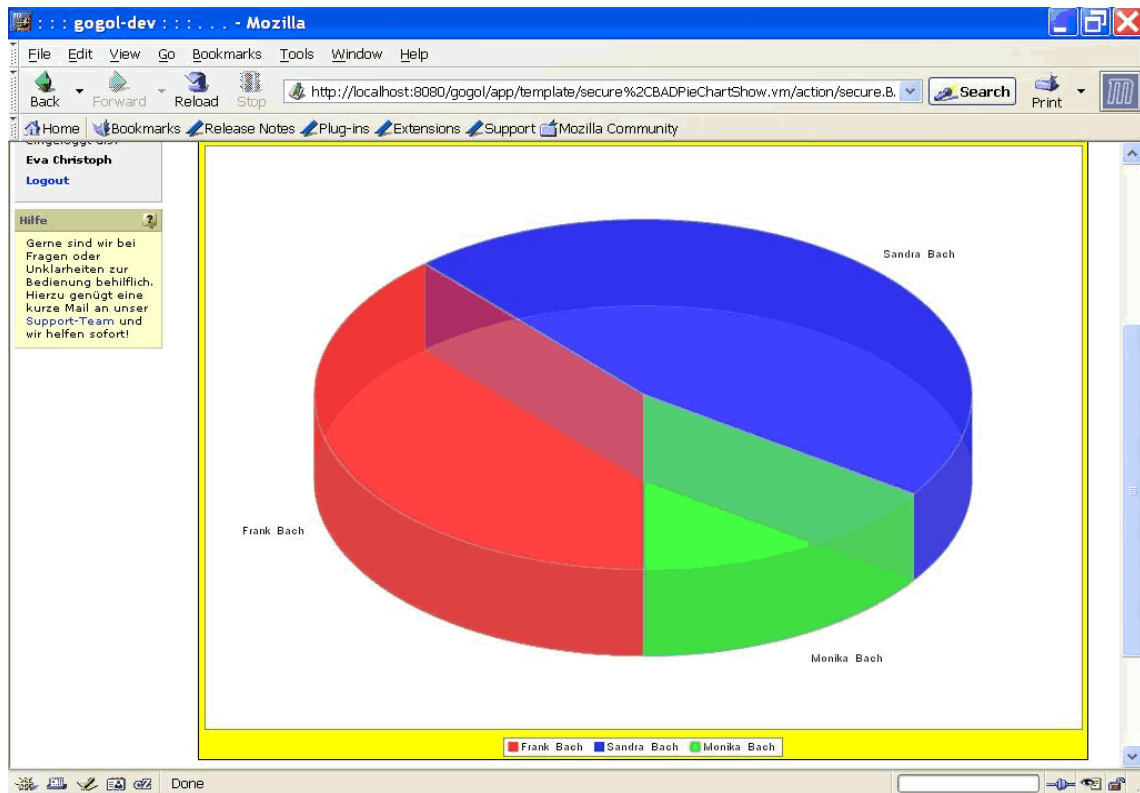
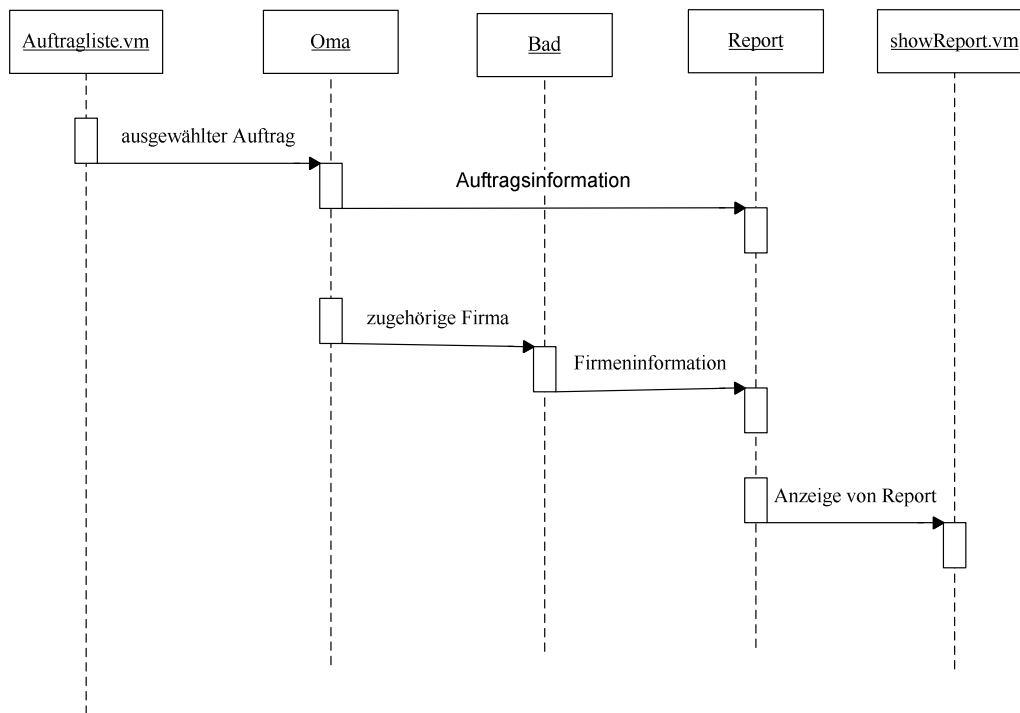


Abbildung 2-24 Das Tortendiagramm von Chart

### 2.3.8 Generierung von Berichten (report)

Die von der Komponente *Report* benötigten Daten müssen ebenfalls mit Hilfe von *bad* und *oma* aus der Datenbank geholt werden.

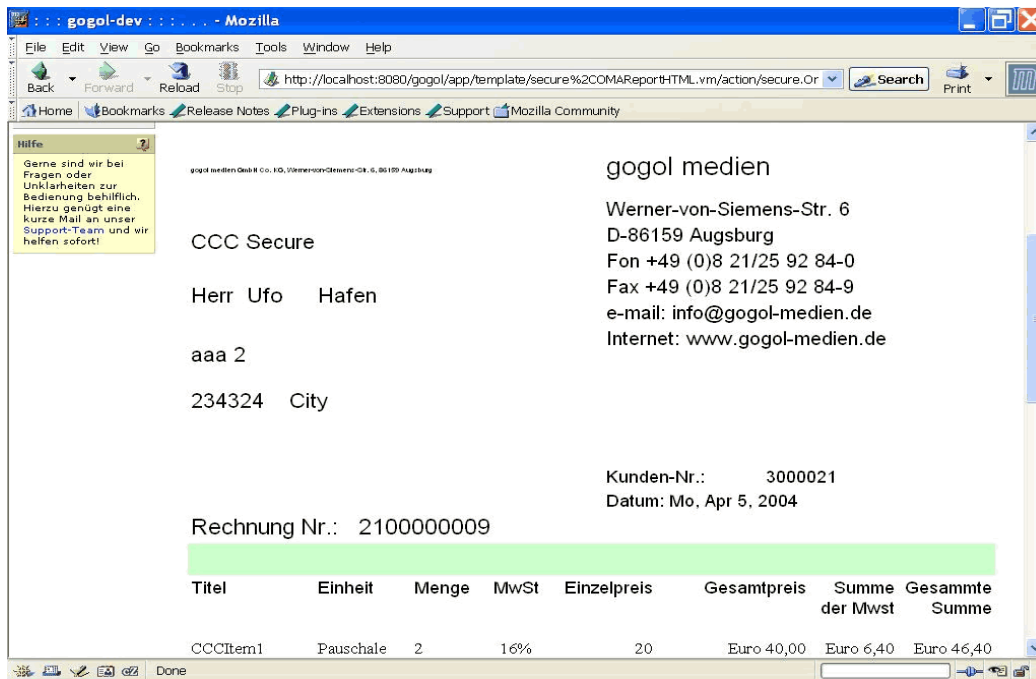
Das in der Abbildung 3-25 angezeigte Diagramm erklärt den Ablauf von der Generierung eines Berichtes.



**Abbildung 2-25 Ablauf von der Komponente Report**

Durch den ausgewählten Auftrag werden die Daten von der betreffenden Firma und dessen Ansprechpartner bestimmt. Dann müssen alle Auftragsitems für diesen Auftrag ausgelesen und temporär gespeichert werden. Beim letzten Schritt werden die Daten nach der Spezifikation von der Komponente Report (siehe Abschnitt 3.2.4.8) geordnet und weitergeleitet. Die Daten werden dann bei Report in `TabelModel`<sup>1</sup> umgespeichert, und nach der Definition der XML-Datei angezeigt. Der Bericht wird dann zuerst in HTML generiert, und gleich in der Web-Browser angezeigt, danach kann der Benutzer noch fordern, dass der Bericht in PDF generiert wird.

<sup>1</sup> `TabelModel` basiert auf `JTable`, die aus dem Packet `javax.swing.JTable` stammt.



**Abbildung 2-26** der generierte Bericht in HTML

Die Abbildung 3-26 zeigt einen Blick von dem generierten Bericht in HTML.

## 2.4 Ausblicke

Kein System ist perfekt, man kann noch viele Sachen bei diesem System verbessern.

Der große Nachteil von dem jetzigen System ist, dass die Komponenten *bad* und *oma* von einander abhängig sind, da die Komponenten *Chart* und *Report* die beiden Komponenten benötigen, und sie liegen jeweils unter *bad* und *oma*, deshalb kann man die Abhängigkeit nicht vermeiden. Ein Lösungsansatz wäre, dass beide Komponenten (*bad* und *oma*) die gleichen Peer-Klassen enthalten, d.h. *bad* enthält z.B. nicht nur alle Peer-Klassen von sich selbst, sondern auch die Peer-Klassen von *oma*, dann sind sie getrennte Teile. Aber es gibt in diesem Fall wieder Coderedundanz.

Es gibt noch große Spielräume mit Velocity, damit das Userinterface noch besser aussieht.

Also vom Ganzen gesehen erfüllt das System schon die vorher definierten Anforderungen. Für das Verfeinern kann man noch vieles machen.

## 3 Synchronisation

### 3.1 Situation/Aufgabenstellung

Gegeben ist eine Access Datenbankstruktur, gefüllt mit Kunden und Auftragsdaten., die auf eine neue, besser angepasste Struktur überspielt werden sollen. Das neue System wird nicht von Anfang an alle Funktionen zur Verfügung stellen, deshalb müssen einige Aufgaben noch im Access System vollzogen werden. In der neuen Lösung werden anfangs nur Berichte und Charts erstellt. Da deshalb beide Datenbanken eine zeitlang parallel im Einsatz sind, soll der die Synchronisation der Daten in beide Richtungen realisiert werden, so dass beide Seiten die gleichen aktuellen Daten aufweisen.

Vorab ein paar Erläuterungen zu Begriffen die in diesem Dokument verwendet werden.

Wenn in diesem Dokument das Wort ‚Kontaktdaten‘ fällt, so ist damit auf der Access Seite die Tabelle ‚Stammdaten‘ gemeint. In Verbindung mit der Postgre Datenbank bezieht sich ‚Kontaktdaten‘ auf die Tabellen ‚SYS\_CONTACT‘, ‚SYS\_CONTACT\_USER‘, ‚SYS\_ADDRESS‘, ‚SYS\_BANK\_ACCOUNT‘ sowie die dazugehörigen Zuordnungstabellen.

Mit ‚Assignmentdaten‘ sind die Access Tabellen ‚Auftrag‘, ‚Anzeige‘, ‚Projekt‘ und ‚Projektposten‘ gemeint bzw. ‚AST\_ASSIGNMENT‘ und ‚AST\_ASSIGNMENT\_ITEM‘ auf der Postgre Seite.

### 3.2 Realisierung

Es wurde entschieden das Synchronisationstool mit PHP4 zu realisieren. PHP bietet einfache Funktionen zum Anbinden von Datenbanken und es lässt sich schnell entwickeln.

Festgelegt wurde, dass einmal täglich in eine bestimmte Richtung synchronisiert wird.

Es gab zwei Ansätze, wie synchronisiert werden sollte:

- Es werden jedes Mal alle Daten übernommen
- Es werden nur die geänderten Daten synchronisiert

Es wurde sich aus Effizienzgründen für die zweite Variante entschieden. Die Synchronisation ist so schneller, da nur die betroffenen Daten ersetzt werden.

#### 3.2.1 Die Datenbankstruktur

Die Datenbankstruktur der Postgre Datenbank wurde in einem Dokument von der Firma gogol medien festgelegt. Es wurde mit dem festgelegtem Design als Grundlage eine Initialisierungsdatei geschrieben, die die Tabellen und Sequenzen der Postgre Datenbank erstellt und auch erste, grundlegende Daten in die Datenbank einträgt.

Im Folgenden wird kurz aufgezählt, welche Daten nach dem CREATE der Tabellen in die Datenbank eingetragen werden. Es wird zuerst die betroffene Tabelle angegeben, gefolgt von den Einträgen:

- SYS\_CONTACT\_CATEGORY  
Kunde, Agentur, Verkäufer
- SYS\_ADDRESS\_TYPE  
Lieferadresse, Rechnungsadresse

- SYS\_PERMISSION  
View, Edit
- SYS\_ROLE  
Autor
- SYS\_GROUP  
Schwaben, Augsburg, Gersthofen, Friedberg, Neusäß
- PUB\_PUBLICATION  
Gersthofen, Friedberg, Neusäß
- AST\_ASSIGNMENT\_TYPE  
Anzeige, Anzeige 6/6, Anzeige 6/6 Abo, Projekt
- AST\_ASSIGNMENT\_STATE  
Auftrag erhalten, in Arbeit, abgeschlossen, Wartung/Hosting, in Akquise
- AST\_MEASURE  
Pauschale, Manntag, Monat, Stück, Stunde, GigaByte, Jahr
- AST\_VAT  
16%, 7%, 0%
- AST\_DOCUMENT\_TYPE  
Auftragsbestätigung, Rechnung, Rechnungskopie, 1. Mahnung, 2. Mahnung, Letzte Mahnung
- AST\_DOCUMENT\_STATE  
offen, erledigt
- SYS\_CONTACT  
Es werden Verkäufer eingetragen.

### 3.2.2 Mapping

Das Mapping stellt die passenden Felder der neuen und alten Datenbank gegenüber. Es ist nicht möglich, alle Felder beider Datenbanken zu mappen, da sich die Datenbankstrukturen teilweise doch sehr unterscheiden.

In den nächsten Abschnitten werden die Felder der Kontaktdaten und der Assignmentdaten, wie sie für die Synchronisation gemappt werden gegenüber gestellt. Die Daten konnten oft nicht immer problemlos eins zu eins übernommen werden. Die genaue Realisierung ist in der Beschreibung der Synchronisationsklassen zu finden.

Das Mapping wurde mit der Firma gogol medien entwickelt.

Die Tabellen zeigen auf der linken Seite die Access Datenbank Felder und auf der rechten Seite die Postgre Felder. Zwischen den beiden Spalten ist in einer Spalte symbolisiert ob und in welche Richtung die Synchronisation läuft:

←,→ Zeigt die Richtung an in welche die Daten übernommen werden

X Diese Felder werden nicht berücksichtigt

#### 3.2.2.1 Kontaktdaten

Die Kontaktdaten in der Access Datenbank liegen alle in der Tabelle ‚Stammdaten‘. Im neuen Design gibt es für die Kontaktdaten die Tabellen ‚SYS\_CONTACT‘, ‚SYS\_CONTACT\_USER‘, ‚SYS\_ADDRESS‘ und ‚SYS\_BANK\_ACCOUNT‘. **Fehler! Verweisquelle konnte nicht gefunden werden.** zeigt diese Zuordnung.

| <b>Access</b>          |   | <b>Postgre</b>  |
|------------------------|---|---|
| <b>Stammdaten</b>      |   |   |
| .Anrede                | ↔ | SYS_CONTACT_USER.salutation                             |
| .Vorname               | ↔ | SYS_CONTACT_USER.firstname                              |
| .Nachname              | ↔ | SYS_CONTACT_USER.lastname                               |
| .Kundennummer          | ↔ | SYS_CONTACT.id  |
| .Firma                 | ↔ | SYS_CONTACT.name1                                       |
| .Firma2                | ↔ | SYS_CONTACT.name2                                       |
| .Strasse               | ↔ | SYS_ADDRESS.street                                      |
| .PLZ                   | ↔ | SYS_ADDRESS.zip_code                                    |
| .Ort                   | ↔ | SYS_ADDRESS.city  |
| .Telefon               | ↔ | SYS_CONTACT_USER.telephone<br>SYS_CONTACT.telephone     |
| .Telefax               | ↔ | SYS_CONTACT.fax<br>SYS_CONTACT_USER.fax                 |
| .Telefon2              | → | SYS_CONTACT_USER.telephone                              |
| .Mobil                 | ↔ | SYS_CONTACT_USER.mobile                                 |
| .WWW                   | ↔ | SYS_CONTACT.internet<br>SYS_CONTACT_USER.internet       |
| .E-Mail                | ↔ | SYS_CONTACT.email<br>SYS_CONTACT_USER.email             |
| .Geburtsdatum          | X |   |
| .Cronologie-Redaktion  | X |   |
| .Chronologie-Anzeigen  | X |   |
| .Zeitstempel-Anzeigen  | X |   |
| .Zeitstempel-Redaktion | X |   |
| .Medium Bekannt        | X |   |
| .Vertriebsnummer       | X |   |
| .Anzahl der Exemplare  | X |   |
| .Auslegeswerpunkt      | X |   |
| .tot                   | ↔ | SYS_CONTACT.flag_active<br>SYS_CONTACT_USER.flag_active |
| .Agentur               | X |   |
| .Auslegeplatz          | X |   |
| .AP-Vtrb Anrede        | X |   |
| .AP-Vtrb Vorname       | X |   |
| .AP-Vtrb Nachname      | X |   |
| .Telefon-Vtrb          | X |   |
| .Ereichbarkeit         | X |   |
| .Branche               | X |   |
| .Kto-Nr                | X |   |
| .BLZ                   | X |   |
| .PasswortMail          | X |   |
| .PasswortIntranet      | X |   |
| .Mailing               | X |   |
| .status                | X |   |
| .keyaccount            | X |   |
| .blz-kunde             | ↔ | SYS_BANK_ACCOUNT.bank_code                              |
| .kto-nr-kunde          | ↔ | SYS_BANK_ACCOUNT.account_number                         |
| .AuslageRegion         | X |   |
| .Zahlungsziel          | ↔ | SYS_CONTACT.term_of_payment                             |
| .ZahlungszielSkonto    | ↔ | SYS_CONTACT.term_of_payment_discount                    |
| .Skonto                | ↔ | SYS_CONTACT.discount                                    |

**Tabelle 1** Mapping Kontaktdaten

### 3.2.2.2 Assignmentdaten

Die Aufträge aus der Tabelle ‚Auftrag‘ und die Projekte aus der Tabelle ‚Projekte‘ werden in dieselbe Tabelle ‚AST\_ASSIGNMENT‘ übernommen. Genauso ist es mit den Tabellen ‚Anzeigen‘ und ‚Projektposten‘. Diese werden in der Tabelle ‚AST\_ASSIGNMENT\_ITEM‘ gespeichert.

Folgende Tabellen aus dem neuen Design sind betroffen:

- AST\_ASSIGNMENT
- AST\_ASSIGNMENT\_ITEM

- AST\_DOCUMENT
- AST\_DOCUMENT2AST\_ASSIGNMENT\_ITEM
- PUB\_EDITION
- PUB\_PUBLICATION

**zeigt die Zuordnung der Tabelle ‚Auftrag‘ zu den Postgre Tabellen, Tabelle 2**  
Mapping Auftrag - Assignment

die ‚Anzeigen‘, **Fehler! Verweisquelle konnte nicht gefunden werden.** die ‚Projekte‘ und **Fehler! Verweisquelle konnte nicht gefunden werden.** die ‚Projektposten‘.

Bei den Assignmentdaten war es schwieriger die Synchronisation in die richtigen Tabellen zu implementieren. Die genaue Realisierung ist wiederum in den Klassenbeschreibungen zu finden.

| <b>Auftrag</b>           |   |  |
|--------------------------|---|--|
| .Kundennummer            | ↔ | AST_ASSIGNMENT.customer_contact_id   |
| .Auftragsnummer          | ↔ | AST_ASSIGNMENT.id  |
| .Format                  | ↔ | AST_ASSIGNMENT_ITEM.ad_format_id   |
| .Farbe                   | ↔ | AST_ASSIGNMENT_ITEM.flag_colour  |
| .Listenpreis             | ↔ | AST_ASSIGNMENT_ITEM.list_price   |
| .Rabatt                  | X |  |
| .Nettopreis              | ↔ | AST_ASSIGNMENT_ITEM.net_price  |
| .Rabattgrund             | X |  |
| .Region                  |   |  |
| .Verkäufer               | ↔ | AST_ASSIGNMENT.salesman_contact_user_id  |
| .AProvision              | ↔ | ASSIGNMENT.salesman_fee  |
| .Malstaffel              | ↔ | AST_ASSIGNMENT.ast_assignment_type_id<br>AST_ASSIGNMENT.title                      |
| .Placierung              | ↔ | AST_ASSIGNMENT_ITEM.placement  |
| .ADatum                  | ↔ | AST_ASSIGNMENT.assignment_date   |
| .Bankeinzug              | ↔ | AST_ASSIGNMENT.bank_account_id   |
| .Placierung_Kundenwunsch | ↔ | AST_ASSIGNMENT_ITEM.placement<br>AST_ASSIGNMENT_ITEM.placement_customer_preference |

**Tabelle 2 Mapping Auftrag - Assignment**

| <b>Anzeige</b>   |   |   |
|------------------|---|---|
| .Auftragsnummer  | ↔ | AST_ASSIGNMENT_ITEM.assignment_id       |
| .Ausgabe         | ↔ | AST_ASSIGNMENT_ITEM.edition_id          |
| .Status          | ↔ | AST_DOCUMENT.document_type_id           |
| .Datum Status    | X |   |
| .Zahlung         | ↔ | AST_DOCUMENT.document_state_id          |
| .ProvStatus      | ↔ | AST_ASSIGNMENT.settlement_date_salesman |
| .Motiv           | X |   |
| .Anzeigenummer   | ↔ | AST_ASSIGNMENT_ITEM.id                  |
| .Zahlungseingang | ↔ | AST_DOCUMENT.actual_value               |

**Tabelle 3 Mapping Anzeige – Assignment Item**

| Projekte             |   |                                    |
|----------------------|---|------------------------------------|
| .Projektnummer       | ↔ | AST_ASSIGNMENT.id                  |
| .Projektbeschreibung | ↔ | AST_ASSIGNMENT.title               |
| .Status-Datum        | X |                                    |
| .Status              | ↔ | AST_DOCUMENT.document_type_id      |
| .Zahlung             | X |                                    |
| .Kundennummer        | ↔ | AST_ASSIGNMENT.customer_contact_id |
| .Volumen             | → | ASSIGNMENT.list_price              |
|                      | ↔ | ASSIGNMENT.net_price               |
| .Fremdleistung       | X |                                    |
| .Dienstleister       | X |                                    |

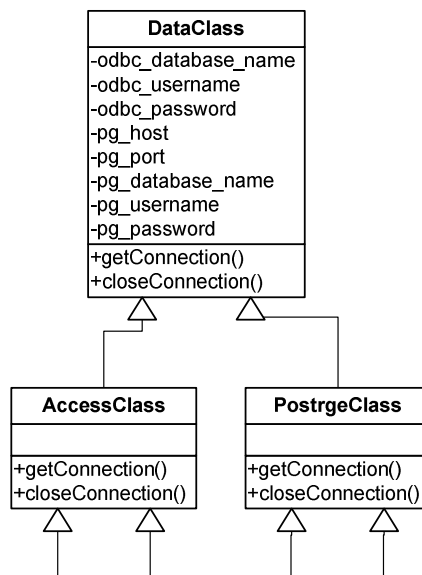
**Tabelle 4** Mapping Projekte - Assignment

| Projektposten    |   |   |
|------------------|---|---|
| .Id              | ↔ | AST_ASSIGNMENT_ITEM.id  |
| .Menge           | ↔ | AST_ASSIGNMENT_ITEM.amount                                      |
| .Einheit         | ↔ | AST_ASSIGNMENT_ITEM.measure_id                                  |
| .Preis/E         | ↔ | AST_ASSIGNMENT_ITEM.list_price<br>AST_ASSIGNMENT_ITEM.net_price |
| .USt             | ↔ | AST_ASSIGNMENT_ITEM.vat_id                                      |
| .Bezeichnung     | ↔ | AST_ASSIGNMENT_ITEM.title                                       |
| .Projektnummer   | ↔ | AST_ASSIGNMENT_ITEM.ast_assignment_id                           |
| .Rechnungsdatum  | ↔ | AST_ASSIGNMENT_ITEM.accounting_date                             |
| .Rechnungsnummer | ↔ | AST_DOCUMENT.invoice_number                                     |

**Tabelle 5** Mapping Projektposten – Assignment Item

### 3.2.3 Klassen

Ausgegangen wird von der Klasse ‚DataClass‘. Diese Klasse stellt die Methoden getConnection() und closeConnection() zur Verfügung, die in den abgeleiteten Klassen ‚AccessClass‘ und ‚PostgreClass‘ datenbankspezifisch überschrieben werden. Die Attribute in ‚DataClass‘ enthalten die Verbindungsdaten zu den Datenbanken (siehe Abbildung 3-1 **Klassenübersicht**).



**Abbildung 3-1** Klassenübersicht Access, Postgre

### 3.2.3.1 Access Datenbank

Eine Instanz der Klasse ‚AccessClass‘ beinhaltet alle für die Synchronisation benötigten Daten aus der Access Datenbank. Abgeleitet von ‚AccessClass‘ werden die Klassen ‚AccessContacts‘ und ‚AccessAssignments‘.

Die Klasse ‚AccessContacts‘ holt sich beim Instanzieren alle benötigten Datensätze aus der Access Tabelle ‚Stammdaten‘. Jeder Datensatz aus ‚Stammdaten‘ wird in einem Objekt der Klasse ‚Stammdaten‘ gespeichert. Das Array ‚array\_stammdaten‘ beinhaltet all diese Stammdaten Datensätze.

Die Aufträge und Projekte werden in ‚AccessAssignments‘ verwaltet. Die Datensätze werden ebenfalls beim Instanzieren der Klasse ‚AccessAssignment‘ aus der Datenbank gelesen und in Objekten der Klassen ‚Auftrag‘, ‚Anzeige‘, ‚Projekt‘ und ‚Projektposten‘ gespeichert. Gespeichert werden diese in den Arrays ‚array\_auftrag‘, ‚array\_anzeige‘, ‚array\_projekt‘ und ‚array\_projektposten‘. Deutlich wird dies in **Fehler! Verweisquelle konnte nicht gefunden werden..**

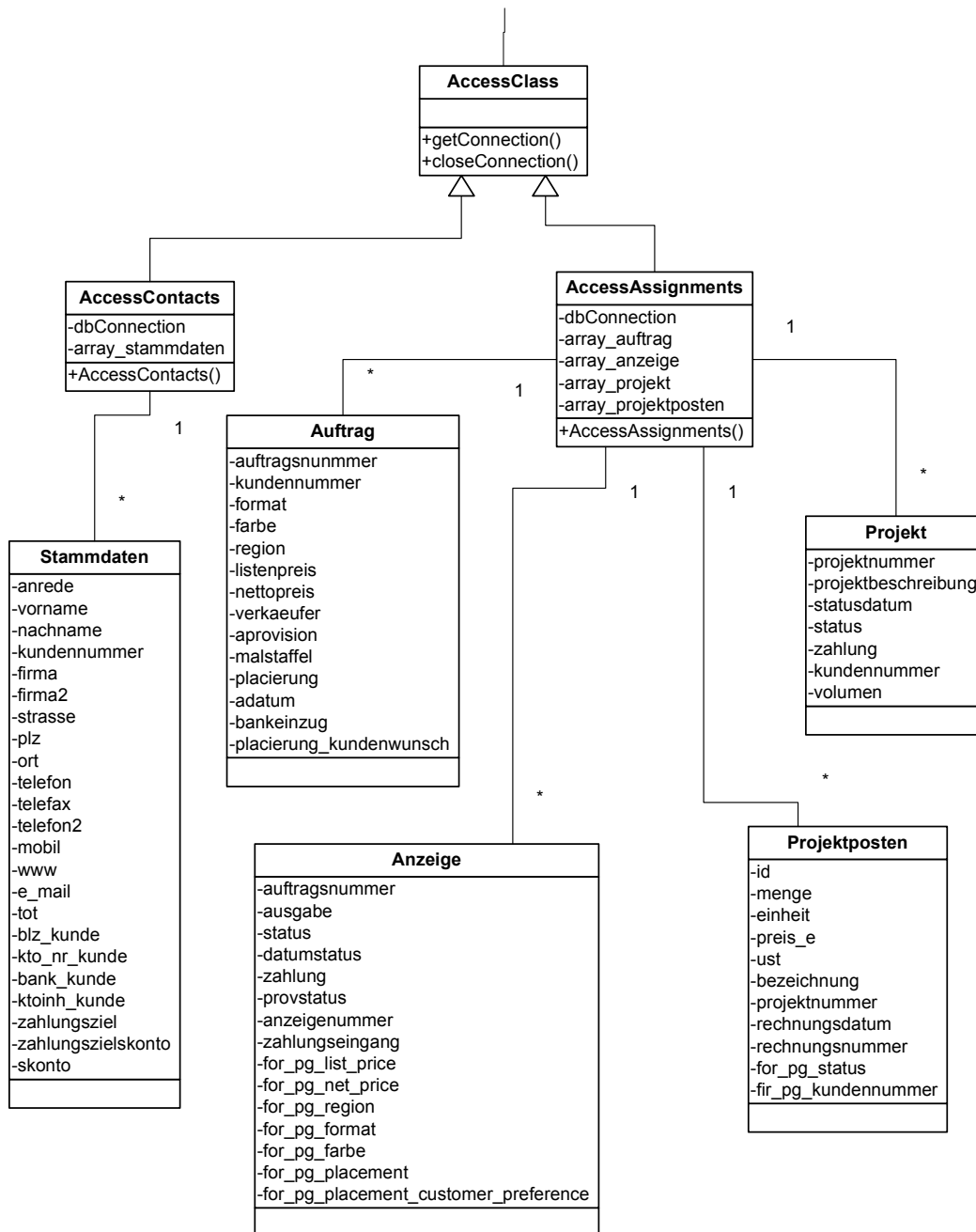


Abbildung 3-2 Klassenübersicht Access

### 3.2.3.2 PostgreSQL Datenbank

‚PostgreAssignments‘ und ‚PostgreContacts‘ sind Unterklassen der Klasse ‚PostgreClass‘. Die Speicherung ist der von Access gleich, nur dass hier eben die Daten aus der Postgre Datenbank geholt und verwaltet werden (siehe Abbildung 3-3 **Klassenübersicht PostgreSQL**). In den Arrays ‚array\_assignment‘ und ‚array\_assignment\_item‘ sind sämtliche Assignmentdaten gespeichert. Die

Kontaktdaten sind in ‚array\_contact‘, ‚array\_contact\_user‘, ‚array\_address‘ und ‚array\_bank\_account‘.

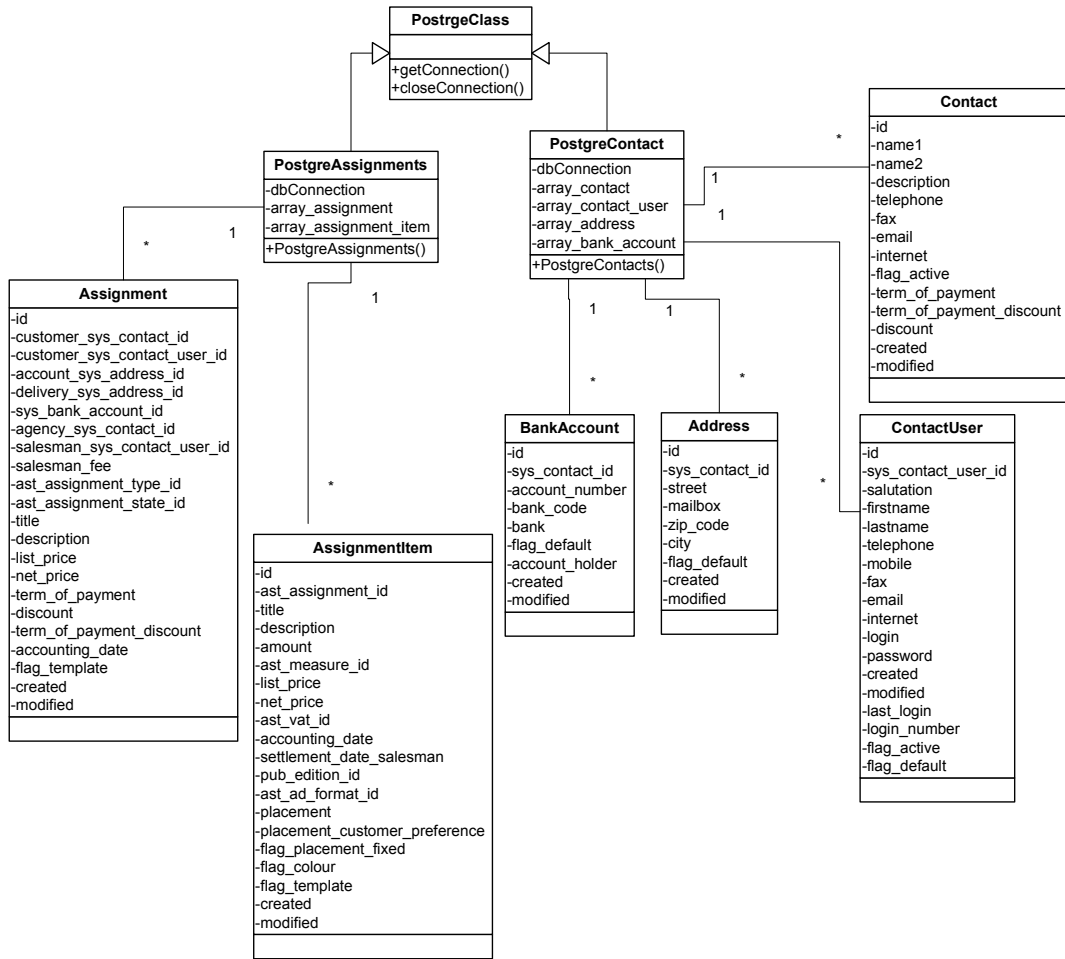


Abbildung 3-3 Klassenübersicht PostgreSQL

### 3.2.3.3 Synchronisationsklassen

Es wurde bisher die Speicherungsstruktur der Datensätze beschrieben. Nun wird das Design der Synchronisation erläutert.

Die Klassen ‚A2P\_sync‘ und ‚P2A\_sync‘ werden von der Klasse ‚SyncClass‘ abgeleitet (siehe Abbildung 3-4 **Klassenübersicht Synchronisation**). Je nach ausgewählter Synchronisationsrichtung wird eine der Klassen instanziiert. ‚A2P\_sync‘ für die Synchronisation von Access nach Postgre und ‚P2A\_sync‘ für die Synchronisation von Postgre nach Access. Die Klassen werden weiter unten genauer beschrieben.

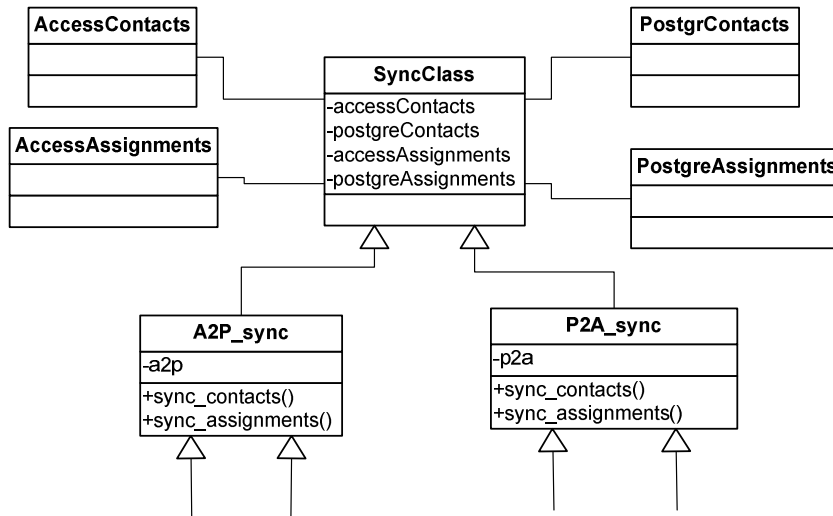


Abbildung 3-4 Klassenübersicht Synchronisation

Die beiden Klassen ‚A2P\_sync‘ und ‚P2A\_sync‘ benötigen weitere Klassen um die Synchronisation durchzuführen (siehe **Fehler! Verweisquelle konnte nicht gefunden werden.**):

- ‚CompareClass‘ beinhaltet Methoden zum Vergleichen der geladenen Daten.
- ‚Access\_2\_Postgre‘ beinhaltet Methoden für die INSERT und UPDATE Statements für das Schreiben in die Postgre Datenbank.
- ‚Postgre\_2\_Access‘ stellt Methoden für das Schreiben in die Access Datenbank zur Verfügung.

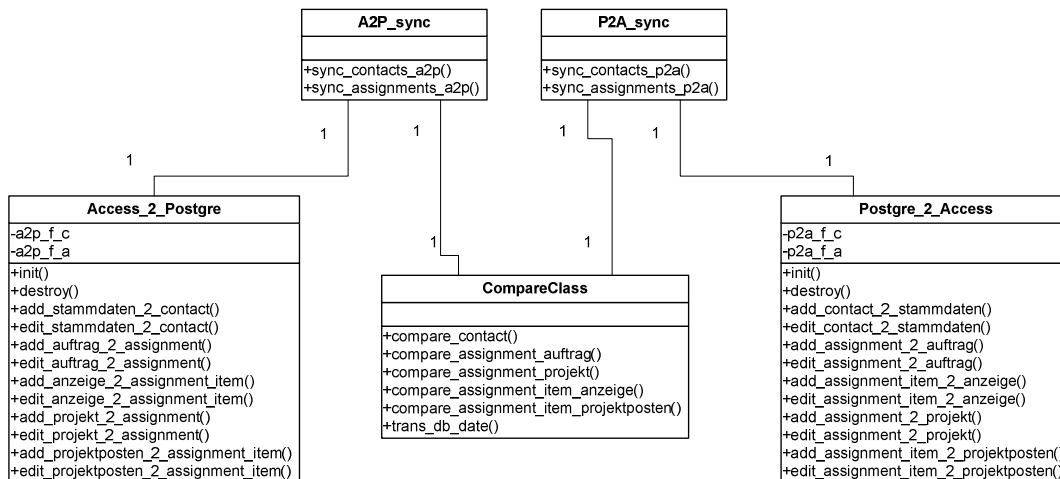


Abbildung 3-5 Klassenübersicht Synchronisation

Als Hilfsfunktionen für ‚Access\_2\_Postgre‘ und ‚Postgre\_2\_Access‘ stehen von der Klasse ‚Filler‘ abgeleitete Klassen ‚A2P\_filler\_contact‘, ‚A2P\_filler\_assignment‘, ‚P2A\_filler\_contact‘ und ‚P2A\_filler\_assignment‘ zur Verfügung. Diese Klassen

enthalten Methoden, die die benötigten Daten für die INSERT und UPDATE Statements holen und für die Datenbanken aufbereiten.

Die Klasse ‚DocumentClass‘ sorgt dafür, dass die im neuen Design benötigten AST\_DOCUMENTs beim Übernehmen einer Anzeige oder eines Projektpostens von Access nach Postgre angelegt werden. **Fehler! Verweisquelle konnte nicht gefunden werden.** zeigt diese Struktur.

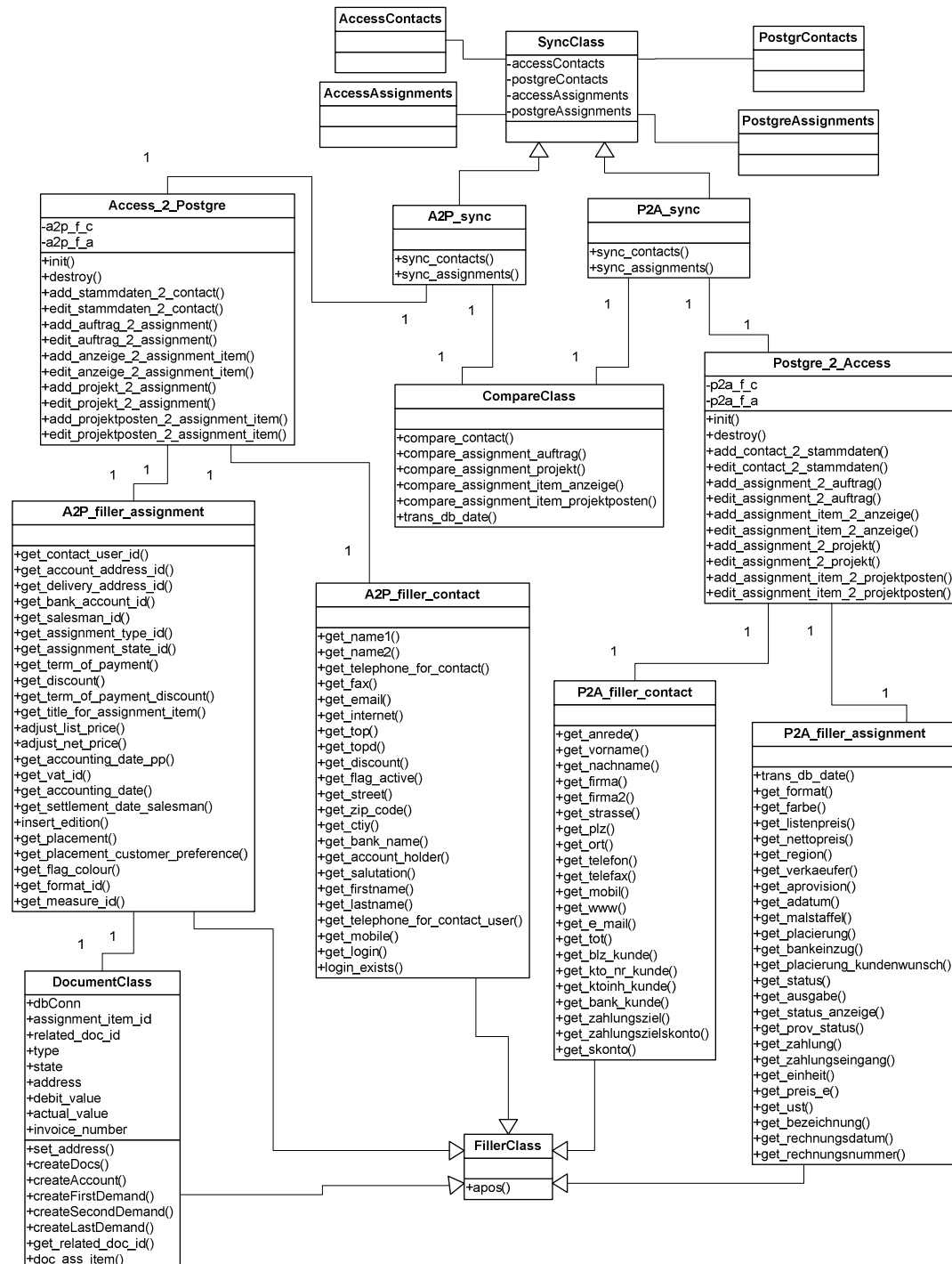


Abbildung 3-6 Klassenübersicht gesamt

## Die Klasse ‚SyncClass‘

Diese Klasse beinhaltet vier Variablen. Die Variablen sind Arrays die alle geladenen Daten aus der Datenbank enthalten und in den Subklassen ausgewertet und verarbeitet werden.

Die in den nachfolgenden Abschnitten erklärten Klassen ‚A2P\_sync‘ und ‚P2A\_sync‘ steuern die Synchronisation. Jede Klasse ist für eine andere Richtung zuständig. Das Vorgehen ist dabei immer gleich:

- Erzeugen der Instanzen für die Kontaktdaten bzw. Assignmentdaten.
- Aufbauen einer Verbindung zur Datenbank, um weitere Daten für das Synchronisieren aus den Datenbanken zu holen.
- Erzeugen einer Instanz der Klasse ‚Access\_2\_Postgre‘ bzw. ‚Postgre\_2\_Access‘ um die neuen oder geänderten Daten in die Zieldatenbank schreiben zu können.
- Erstellen einer Instanz der Klasse ‚CompareClass‘ um die Vergleiche der Daten durchzuführen.

## Die Klasse ‚A2P\_sync‘

Die Methode ‚sync\_contacts()‘ regelt die Synchronisation der Kontaktdaten, die Methode ‚sync\_assignments()‘ die der Auftrags- und Projektdaten, sowie die Anzeigen und Projektposten.

### Methode ‚sync\_contacts()‘

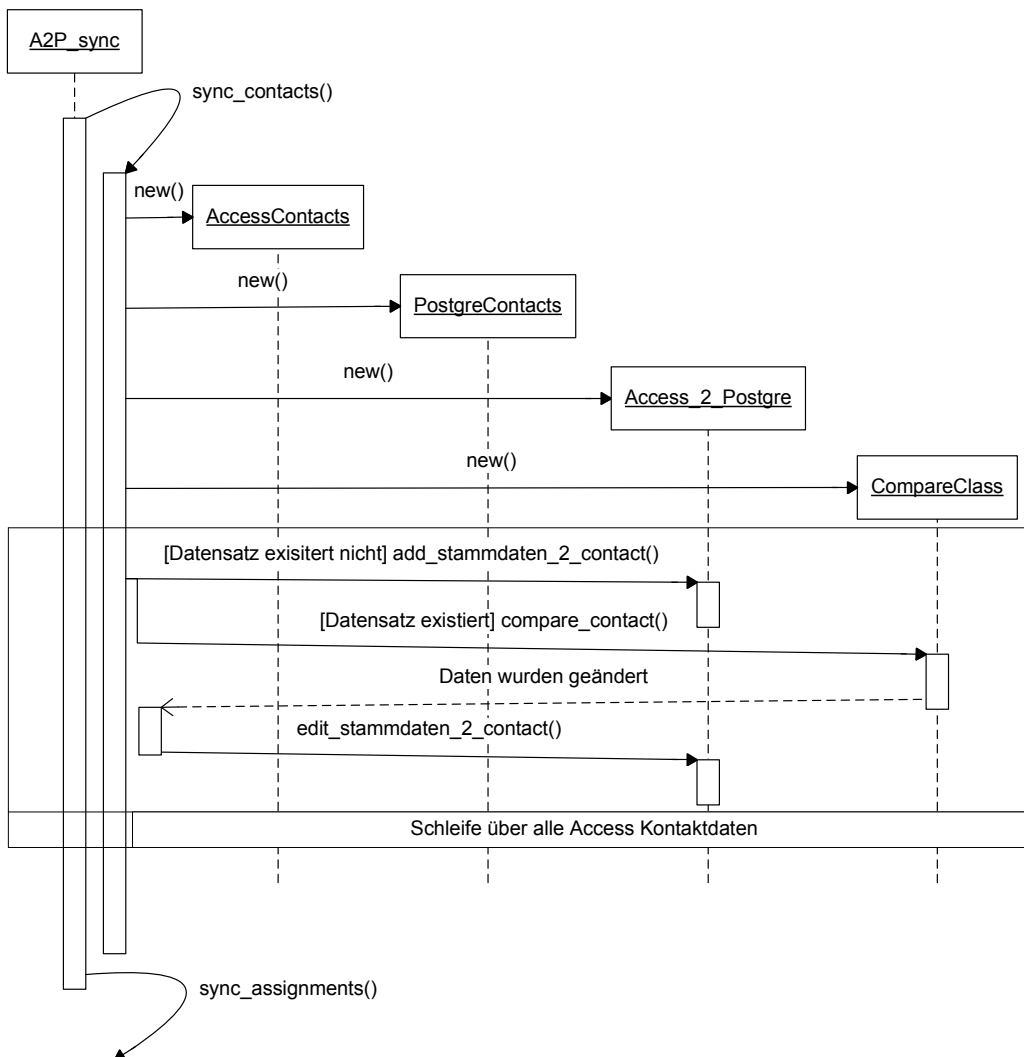
\$accessContacts ist ein Array mit allen Kontaktdaten aus der Access Datenbank. \$postgreContacts beinhaltet alle Kontaktdaten aus der Postgre Datenbank.

Es wird nun zu jedem Kontaktdaten-Datensatz aus ‚array\_stammdaten‘ der passende Kontaktdaten-Datensatz aus ‚array\_contact‘ gesucht. Verglichen werden Stammdaten.Kundennummer und die SYS\_CONTACT.id.

Sollte er nicht gefunden werden, so wird der Kontaktdaten-Datensatz mittels der Methode ‚add\_stammdaten\_2\_contact()‘ aus der ‚Access\_2\_Postgre‘ Klasse in die Postgre Datenbank übernommen.

Sollte die Kundennummer im Feld ‚SYS\_CONTACT.id‘ gefunden werden, so werden die Daten mittels der Methode ‚compare\_contact()‘ (aus ‚CompareClass‘) auf Änderungen untersucht. Es werden sämtliche zu vergleichende Daten der Funktion übergeben. Der Rückgabewert ist ein Array aus Flags, die die geänderten Daten widerspiegeln. Sollten sich Daten geändert haben, so wird das Array mit den Flags der Funktion ‚edit\_stammdaten\_2\_contact()‘ übergeben, die sich um den Angerich der Kontaktdaten kümmert.

Das Sequenzdiagramm **Fehler! Verweisquelle konnte nicht gefunden werden.** zeigt das Verfahren.



**Abbildung 3-7** Sequenzdiagramm Synchronisation der Kontakte

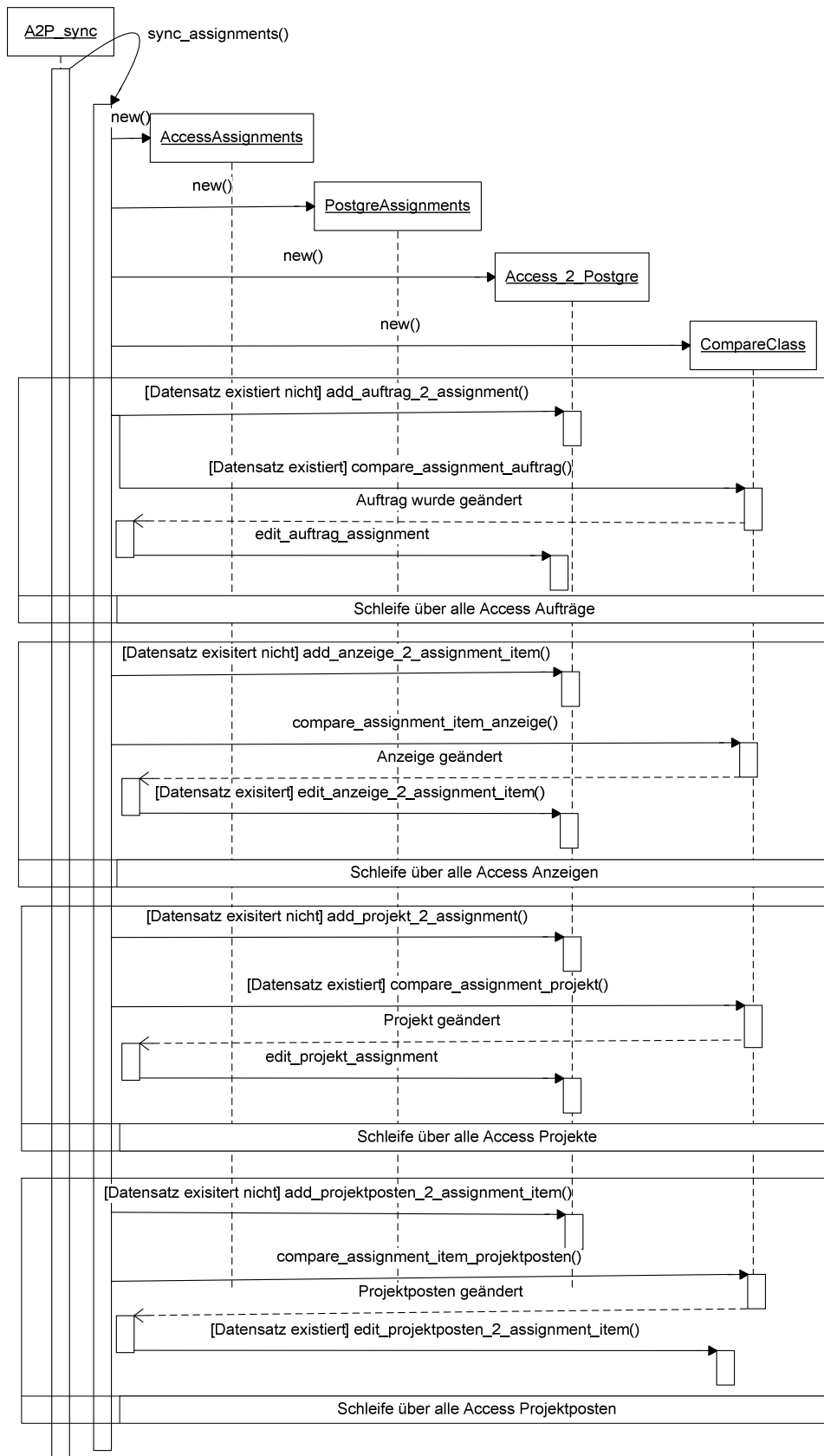
**Methode ‚sync\_assignments()‘**

Die Einträge aus ‚array\_auftrag‘ werden in ‚array\_assignment‘ gesucht und, wie bei den Kontakten, neu angelegt oder verglichen. Hier werden Auftrag.Auftragsnummer und SYS\_ASSIGNMENT.id verglichen. Die Anzeigen aus ‚array\_anzeige‘ werden über die Anzeige.Anzeigennummer in ‚array\_assignment\_item‘ (AST\_ASSIGNMENT\_ITEM.id) gesucht.

Die Zuordnung der Projekte ist wie folgt, wobei sich hier die IDs unterscheiden (siehe unten):

- ‚array\_projekte‘ – ‚array\_assignment‘
- ‚array\_projektposten‘ – ‚array\_assignment\_item‘

Das Sequenzdiagramm **Fehler! Verweisquelle konnte nicht gefunden werden.** zeigt den Ablauf.



**Abbildung 3-8** Sequenzdiagramm Synchronisation der Auftragsdaten

Ein Sonderfall sind die Projekte und die Projektposten. Diese müssen gesondert behandelt werden, da sie im neuen Datenbankdesign wie die Aufträge auf AST\_ASSIGNMENT bzw. die Projektposten wie die Anzeigen auf AST\_ASSIGNMENT\_ITEM abgebildet werden. In der Access Datenbank wurden Aufträge und Projekte in getrennten Tabellen verwaltet und konnten somit auch die gleiche ID (Auftragsnummer, Projektnummer bzw. Anzeigennummer, Projektposten-ID) haben. Im neuen Design müssen diese unterschiedliche IDs bekommen, da sie sich in der gleichen Tabelle befinden.

Um die eindeutige Zuordnung der Aufträge, Projekte, Anzeigen und Projektposten zu den in die Postgre Datenbank Übernommenen Assignmentdaten zu gewährleisten, bekommen die Projekte und Projektposten in der PostgreSQL Datenbank eine neue ID zugewiesen.

Die Auftragsnummern liegen zwischen ca. -2.112.793.001 bis ca. 2.069.589.406, die Projektnummern bei ca. 12 bis ca. 40.

Beim Übernehmen eines Projektes von alt auf neu wird auf die Projektnummer 2.070.000.000 addiert. Die automatische Sequenz von ASSIGNMENT.id beginnt bei 2.100.000.000. So ist es möglich die AST\_ASSIGNMENTS bei einer Synchronisation den jeweils alten Projekten zuzuordnen:

- Aufträge haben IDs zwischen ca. -2.112.793.001 bis ca. 2.069.589.406.
- Die von der Access Datenbank übernommenen Projekte haben IDs zwischen 2.070.000.000 und 2.100.000.000.
- Alle in der Postgre Datenbank neu angelegten Projekte und Aufträge haben IDs ab 2.100.000.000, und sind somit auf beiden Seiten eindeutig.

Das gleiche Problem gibt es bei den Projektposten und den Anzeigen. Die Anzeigennummern liegen zwischen ca. 46 und ca. 1900 die Projektposten-IDs zwischen ca. 5 und ca. 120. Hier werden auf die Projektposten-Ids 30.000 addiert. Die automatische Sequenz beginnt bei 40.000.

### **Die Klasse ‚P2A\_sync‘**

Die Methoden ‚sync\_contacts()‘ und ‚sync\_assignments()‘ synchronisieren die Daten in die Rückrichtung. Der Ablauf entspricht dem in der Klasse ‚A2P\_sync‘. Es wird das Array ‚array\_contact‘ durchlaufen und die entsprechenden Datensätze in dem Array ‚array\_stammdaten‘ gesucht, gegebenenfalls neu angelegt, verglichen und angepasst.

Bei den Assignmentdaten ist der Ablauf der gleiche. Die Arrays ‚array\_assignment‘ und ‚array\_assignment\_item‘ werden durchlaufen und die entsprechenden Daten in ‚array\_auftrag‘, ‚array\_anzeige‘, ‚array\_projekte‘ und ‚array\_projektposten‘ gesucht. Zu beachten ist hier, dass die Projekte und Projektposten andere ID's besitzen als in der Access Datenbank. Dies wird berücksichtigt, indem auf Projektnummern und Projektposten oben beschriebene Offset addiert wird. Somit können auch die übernommenen Projekte synchronisiert werden. Die in Postgre neu angelegten Projekte besitzen in beiden Datenbanken die gleiche ID.

### **Die Klasse ‚CompareClass‘**

Diese Klasse stellt Methoden für den Vergleich der Daten zur Verfügung. Der Input der Methoden besteht immer aus allen, für den Vergleich benötigten Daten. Es wird

aus den übergebenen Daten geprüft, ob sich Datensätze geändert haben. Die Zuordnung wurde im Mapping beschrieben.

In jeder Methode werden Flags für Änderungen gesetzt („true“ für eine Änderung und „false“, falls es keine Änderung gab). Diese Flags werden als Array zurückgegeben. In dem Array wird ein zusätzliches Flag „\$changes“ gespeichert, das angibt ob sich überhaupt Daten geändert haben.

#### **compare\_contact(\$contact, \$array\_contact\_user, \$array\_address, \$array\_bank\_account, \$stammdaten)**

Diese Methode vergleicht Stammdaten aus Access mit den Kontaktdaten aus Postgre. Die folgenden Flags geben an, ob sich Daten geändert haben:

- \$c\_changed - der Kontakteintrag in SYS\_CONTACT hat sich geändert
- \$cu\_new - zu dem Kontaktdateneintrag gibt es einen neuen (default-) Ansprechpartner
- \$cu\_changed - der default SYS\_CONTACT\_USER hat sich geändert
- \$address\_new - der Kontakt hat eine neue (default-) Adresse
- \$address\_changed - der default Adresseintrag hat sich geändert
- \$bank\_new - der Kontakteintrag hat eine neue Bankverbindung
- \$bank\_changed - der default SYS\_BANK\_ACCOUNT hat sich geändert
- \$changes

#### **compare\_assignment\_auftrag(\$auftrag, \$assignment, \$array\_anzeige, \$array\_ai)**

Hier werden die Aufträge mit den entsprechenden ASSIGNMENTS verglichen.

Arrays:

- \$az\_index - hier sind alle Anzeigennummern zu diesem Auftrag enthalten
- \$ai\_index - hier sind alle ASSIGNMENT\_ITEM.id's zu diesem ASSIGNMENT enthalten

Die folgenden Flags geben wieder die Änderungen an:

- \$check\_4\_contact\_user - es ist kein SYS\_CONTACT\_USER eingetragen sein
- \$check\_4\_address - wie \$check\_4\_contact\_user nur mit Adresse
- \$check\_4\_bank - wie \$check\_4\_contact\_user nur mit Bank
- \$salesman\_fee\_changed - Die Provision der Verkäufers hat sich geändert
- \$assignment\_date\_changed - Das Assignmentdatum hat sich geändert
- \$changes

#### **compare\_assignment\_projekt(\$projekt, \$assignment, \$array\_pp, \$array\_ai, \$p\_dbConn, \$a\_dbConn)**

Es werden AST\_ASSIGNMENTS mit den Projekten verglichen.

Arrays:

- pp\_index - Array mit allen Projektposten -IDs zu diesem Projekt
- ai\_index - Array mit allen ASSIGNMENT\_ITEM.id zu diesem ASSIGNMENT

Flags:

- \$check\_4\_contact\_user - es ist kein SYS\_CONTACT\_USER eingetragen
- \$check\_4\_address - wie \$check\_4\_contact\_user nur mit Adresse
- \$title\_changed - Projektbeschreibung hat sich geändert
- \$list\_price\_changed - Volumen / list\_price hat sich geändert
- \$state\_changed - Proejtstatus, AST\_ASSIGNMENT\_STATE hat sich geändert
- \$changes

**compare\_assignment\_item\_anzeige**(\$anzeige, \$assignment\_item, \$array\_auftrag, \$array\_assignment, \$p\_dbConn)

Variablen:

- \$auftrag\_index - der zu dieser Anzeige gehörende Auftrag
- \$assignment\_index - der zu diesem ASSIGNMENT\_ITEM gehörende ASSIGNMENT

Flags: (AI steht für AST\_ASSIGNMENT\_ITEM)

- \$list\_price\_changed - Auftrag.Listenpreis – AI.list\_price
- \$net\_price\_changed - Auftrag.Nettopreis – AI.net\_price
- \$settlement\_date\_salesman\_changed - Anzeige.ProvStatus – AI.settlement\_date\_salesman
- \$edition\_changed - Anzeige.Ausgabe - AI.edition
- \$region\_changed - Auftrag.Region – AI.edition / PUBLICATION.name
- \$format\_changed - Auftrag.Format – AI.ad\_format\_id
- \$placement\_changed - Auftrag.Placierung – AI.placement, Auftrag.Palcierung\_Kundenwunsch – AI.placement\_customer\_preference
- \$colour\_changed - Auftrag.Farbe – AI.flag\_colour

**compare\_assignment\_item\_projektposten**(\$projektposten, \$assignment\_item, \$array\_projekt, \$array\_assignment, \$p\_dbConn)

Variablen:

- \$projekt\_index - Projekt zu dem Projektposten
- \$assignment\_index - ASSIGNMENT zu diesem ASSIGNMENT\_ITEM

Flags (PP = Projektposten, AI = AST\_ASSIGNMENT\_ITEM):

- \$title\_changed - PP.Bezeichnung – AI.title
- \$amount\_changed - PP.Menge – AI.amount
- \$measure\_changed - PP.Einheit – AI.measure\_id
- \$list\_price\_changed - PP.preis/e – AI.list\_price
- \$vat\_changed - PP.USt – AI.vat\_id
- \$accounting\_date\_changed - PP.Rechnungsdatum – AI.accounting\_date
- \$invoice\_number\_changed - PP.Rechnungsnummer – Document.invoice\_number

## **Die Klasse ‚Access\_2\_Postgre‘**

Diese Klasse beinhaltet alle Statements für die INSERTs und UPDATEs nach Postgre. Die ‚add-‘ Funktionen fügen neue Datensätze in die Postgre Datenbank hinzu. Die ‚edit-‘ Funktionen bekommen den Rückgabewert der Vergleichsfunktion (Array aus Flags), werten die Flags in diesem Array aus und führen dementsprechend die UPDATE und INSERT Statements aus. Es werden auch die Zuordnungstabellen, wie SYS\_CONTACT\_CATEGORY2SYS\_CONTACT, mitberücksichtigt.

Oft werden in Postgre Datums eingetragen, die in Access nicht vorhanden sind. In diesen Fällen wird immer der ‚01.01.1970‘ eingetragen.

### **add\_stammdaten\_2\_contact()**

Fügt ein Stammdaten-Datensatz, der noch nicht in den Kontaktdaten der Postgre Datenbank existiert dort ein. Die Daten dafür werden über die Methoden der Filler Klasse ‚A2P\_filler\_contact‘ ermittelt.

Betroffen ist immer die Tabelle ‚SYS\_CONTACT‘ und je nach Vollständigkeit der Access-Kontaktdaten die Tabellen ‚SYS\_CONTACT\_USER‘, ‚SYS\_ADDRESS‘ und ‚SYS\_BANK\_ACCOUNT‘.

### **edit\_stammdaten\_2\_contact()**

Vorgehen, falls Flag ‚true‘ ist:

- \$contact\_changed  
Es werden alle Daten, die den ‚SYS\_CONTACT‘ Eintrag betreffen mittels den Filler Funktionen ermittelt und ein UPDATE ausgeführt.
- \$contact\_user\_new  
Es wurde inzwischen ein Ansprechpartner in die Access Datenbank eingetragen und dieser wird nun in der Postgre Datenbank Tabelle ‚SYS\_CONTACT\_USER‘ angelegt.
- \$contact\_user\_changed  
Der default Ansprechpartner hat sich geändert und die Daten werden geändert.
- \$address\_new  
Es wurde inzwischen eine Adresse in der Access Datenbank eingefügt, die nun in die Postgre Datenbank übernommen wird. Es werden auch die entsprechenden Zuordnungstabellen gefüllt.
- \$address\_changed  
Die Adresse hat sich geändert und wird synchronisiert.
- \$bank\_account\_changed  
Die Bankverbindung hat sich geändert und es wird eine neue default-Bankverbindung angelegt.

### **add\_auftrag\_2\_assignment()**

Fügt einen Auftrag in die Tabelle ‚AST\_ASSIGNMENT‘ hinzu. Die Werte für das INSERT Statement werden über die Funktionen der Klasse ‚A2P\_filler\_assignment‘ ermittelt.

Besonderheiten:

- Der Wert des Feldes ‚accounting\_date‘ ist der Wert aus ‚assignment\_date‘, (adatum) plus 14 Tage.

### **edit\_auftrag\_2\_assignment()**

Vorgehen, falls Flag ‚true‘ ist:

- \$check\_4\_contact\_user  
Es wird geprüft ob inzwischen ein default ‚SYS\_CONTACT\_USER‘ zu dem ‚SYS\_CONTACT‘ des ‚AST\_ASSIGNMENT‘ existiert. Falls ja, wird der ‚SYS\_CONTACT\_USER‘ hier als Ansprechpartner in das Feld AST\_ASSIGNMENT.customer\_sys\_contact\_user\_id eingetragen.
- \$check\_4\_address  
wie \$check\_4\_contact\_user, nur für die default-Adresse. Die Adresse wird als Liefer- und Rechnungsadresse eingetragen.

- \$check\_4\_bank\_account  
wie \$check\_4\_contact\_user, nur für die Bankverbindung. Die neue Bankverbindung wird in das Feld AST\_ASSIGNMENT.bank\_account\_id eingetragen.
- \$salesman\_fee\_changed  
Der neue Wert wird übernommen
- \$assignment\_date\_changed  
Das Datum wird angepasst.

### **add\_anzeige\_2\_assignment\_item()**

Hier werden neue Anzeigen in die Tabelle ‚AST\_ASSIGNMENT\_ITEM‘ übernommen.

Besonderheiten:

- Einige Werte hier stehen in der Tabelle Auftrag (Farbe, Format, Placierung,...). Von dort werden auch die Daten übernommen.
- Der Wert in Auftrag.Listenpreis hat eine etwas andere Bedeutung als der in AST\_ASSIGNMENT.list\_price. In der Access Datenbank steht dort der Preis für eine Anzeige und in Postgre steht der Gesamtpreis aller dazugehörigen AST\_ASSIGNMENT\_ITEMS. Die Funktionen adjust\_net\_price() und adjust\_list\_price() kümmern sich darum und errechnen die Preise für die Felder AST\_ASSIGNMENT.list\_price bzw.net\_price bei jedem neu angelegtem AST\_ASSIGNMENT\_ITEM neu.
- Wurde eine Anzeige übernommen, werden automatisch auch die dazugehörigen Dokumente angelegt. Der Typ der Dokumente ist abhängig ob die Rechnung schon bezahlt wurde oder nicht.

### **edit\_anzeige\_2\_assignment\_item()**

Vorgehen, falls Flag ‚ture‘ ist:

- list\_price\_changed  
Der Listenpreis wird geändert und adjust\_list\_price() ausgeführt.
- net\_price\_changed  
Wie list\_price nur für net\_price
- settlement\_date\_salesman\_changed  
Das Datum wird ausgetauscht.
- edition\_changed  
Sollte die neue Edition noch nicht existieren, so wird diese in die Tabelle ‚PUB\_EDITION‘ eingetragen. (Die dazugehörige PUB\_PUBLICATION muss schon in der Datenbank existieren)
- format\_changed  
Das neue Format wird ermittelt und eingetragen.
- placement\_changed  
Es wird die Placierung sowie der Kundenwunsch aktualisiert.
- colour\_changed  
Die Farbe wird aktualisiert.

### **add\_projekt\_2\_assignment()**

Es wird ein Projekt von der Access Datenbank in die Postgre Datenbank übernommen.

Besonderheiten:

- Die ID wird mit 2.070.000.000 addiert, damit es keine Kollisionen mit den Aufträgen gibt.

### **edit\_projekt\_2\_assignment()**

Projektnummern werden wieder mit 2.070.000.000 addiert, damit die Zuordnung stimmt.

Flags:

- check\_4\_contact\_user  
Wie in Auftrag.
- check\_4\_address  
Wie in Auftrag.
- title\_changed  
Der Titel wird geändert.
- list\_price\_changed  
Der list\_price wird angepasst.
- state\_changed  
Der Status des Projektes hat sich geändert und wird angepasst.

### **add\_projektposten\_2\_assignment\_item()**

Der Projektposten wird in die Tabelle ‚AST\_ASSIGNMENT\_ITEM‘ übernommen.

Besonderheiten:

- Die Id des Projektposten wird mit 30.000 addiert, um Kollisionen mit den Anzeigen zu vermeiden. Die Projektnummer wird gleichermaßen angepasst.
- Es werden die Dokumente zu dem gerade angelegten AST\_ASSIGNMENT\_ITEMS erzeugt.

### **edit\_projektposten\_2\_assignment\_item()**

Vorgehen, falls Flag ‚true‘ ist:

- title\_changed  
Der Titel wird geändert
- amount\_changed  
Die Menge wird angepasst
- measure\_changed  
Die Einheit wird synchronisiert
- list\_price\_changed  
list\_price wird angepasst
- vat\_changed  
VAT wird angepasst
- accounting\_date\_changed  
Das Rechnungsdatum wird geändert
- invoice\_number\_changed  
Die Rechnungsnummer im zugehörigen ‚AST\_DOCUMENT‘ wird geändert

### **Die Klasse ‚Postgre\_2\_Access‘**

Diese Klasse beinhaltet alle Statements für die INSERTs und UPDATEs nach Access. Der Aufbau und Ablauf ist gleich dem von ‚Access\_2\_Postgre‘

### **add\_contact\_2\_stammdaten()**

Fügt einen neuen Stammdaten Datensatz In die Access Datenbank ein. Die Werte werden über Filler Funktionen aus der Klasse ‚P2A\_filler\_contact‘ geholt.

### **edit\_contact\_2\_stammdaten()**

Flags:

- \$contact\_changed
- \$contact\_user\_changed
- \$address\_changed
- \$bank\_account\_changed

Sollte eines der Flags true sein, so wird der ganze Stammdaten Datensatz aktualisiert.

### **add\_assignment\_2\_auftrag()**

Es wird ein neuer Auftrag in der Access Datenbank angelegt. Die Werte werden über die Filler Funktionen der Klasse ‚P2A\_filler\_assignment‘ ermittelt.

### **edit\_assignment\_2\_auftrag()**

Vorgehen, falls Flag ‚ture‘ ist:

- \$salesman\_fee\_changed  
Aprovision wird aktualisiert
- \$assignment\_date\_changed  
ADatum wird aktualisiert
- \$check\_4\_bank\_account  
Es wird geprüft, ob inzwischen eine Bankverbindung existiert und diese wird gegebenenfalls übernommen.

### **add\_assignment\_item\_2\_anzeige()**

Es wird eine neue Anzeige in der Access Datenbank angelegt.

Besonderheiten:

- Sollte sich die PUBLICATION der EDITON geändert haben, so wird die Region des Auftrages geändert.

### **edit\_assignment\_item\_2\_anzeige()**

Vorgehen, falls Flag ‚true‘ ist:

- \$list\_price\_changed  
Listenpreis des Auftrages wird angepasst
- \$net\_price\_changed  
Nettopreis des Auftrags wird angepasst
- \$settlement\_date\_salesman\_changed  
ProvStatus der Anzeige wird angepasst
- \$edition\_changed  
Die Ausgabe der Anzeige und gegebenenfalls wird die Region des Auftrages aktualisiert

- \$format\_changed  
Das Format in Auftrag wird geändert
- \$placement\_changed  
Placierung und Placierung\_Kundenwunsch in Auftrag wird aktualisiert.
- \$colour\_changed  
Farbe in Auftrag wird angepasst.

#### **add\_assignment\_2\_projekt()**

Es wird ein neues Projekt aus der Postgre Datenbank in die Access Datenbank übernommen.

#### **edit\_assignment\_2\_projekt()**

Vorgehen, falls Flag 'true' ist:

- \$title\_changed  
Projektbeschreibung wird angepasst.
- \$list\_price\_changed  
Volumen wird angepasst
- \$state\_changed  
der Status des Projektes wird aktualisiert.

#### **add\_assignment\_item\_2\_projektposten()**

Es wird ein neuer Projektposten aus der Postgre Datenbank übernommen.

#### **edit\_assignment\_item\_2\_projektposten()**

Vorgehen, falls Flag 'true' ist:

- \$title\_changed  
Bezeichnung wird angepasst
- \$measure\_changed  
die Einheit wird angepasst
- \$list\_price\_changed  
Preis/Einheit wird angepasst
- \$vat\_changed  
USt wird angepasst
- \$accounting\_date\_changed  
Rechnungsdatum wird angepasst
- \$invoice\_number\_changed  
Rechnungsnummer wird angepasst

### **Die Klasse ‚FillerClass‘**

Die von der Klasse ‚FillerClass‘ abgeleiteten Klassen ‚A2P\_Filler\_assignment‘, ‚A2P\_filler\_contact‘, ‚P2A\_filer\_assignment‘ und ‚P2A\_filler\_contact‘ stellen Methoden zur Verfügung, die die jeweils benötigten Daten für die INSERT und UPDATE Statements ermitteln und auch gleich das richtigen Format liefern.

‚FillerClass‘ enthält die Funktion apos(\$value), die prüft, ob ein String ein Hochkomma enthält. Falls ja, wird dieses durch zwei Hochkommata ersetzt, damit der String problemlos in die Datenbank eingefügt werden kann.

In den nächsten Abschnitten werden die Funktionen der Filler Funktionen etwas näher beschrieben, wobei nur auf die Besonderheiten eingegangen wird.

Die Filler Funktionen sind meist recht einfach:

- Sie Prüfen ein übergebenen Wert auf Leerheit und liefern gegebenenfalls den String ‚NULL‘ für das SQL Statement zurück.
- Sie holen den benötigten Wert aus der Datenbank und passen es für die Datenbank SQL Statements an. Was geholt wird ist leicht aus den Namen der Methoden ablesbar.

## **Die Klasse ‚A2P\_filler\_assignment‘**

Methoden für INSERT und UPDATE Statements der Assignmentdaten in Richtung Postgre.

- `get_contact_user_id($dbConn,$contact_id)`  
Ermittelt die default SYS\_CONTACT\_USER.id der übergebenen SYS\_CONTACT.id
- `get_account_address($dbConn,$contact_id)`  
Ermittelt die default SYS\_ADDRESS.id vom Typ Rechnungsadresse
- `get_delivery_address($dbConn,$contact_id)`  
Ermittelt die default SYS\_ADDRESS.id vom Typ Lieferadresse
- `get_bank_account($dbConn,$bankeinzug,$contact_id)`  
Sollte \$bankeinzug ‚true‘ sein, so wird die default SYS\_BANK\_ACCOUNT.id ermittelt
- `get_salesman_id($dbConn,$verkaeufel)`  
Über den Nachname in \$verkaeufel wird die SYS\_CONTACT\_USER.id des Verkäufers ermittelt und zurückgegeben. Bei mehreren Treffern wird der erste zurückgegeben, bei keinem Treffer NULL.
- `get_assignment_type_id($dbConn,$malstaffel)`  
Über den Wert in \$malstaffel wird die SYS\_ASSIGNMENT\_TYPE.id ermittelt
- `get_assignment_state_id($dbConn,$status)`  
Es wird die AST\_ASSIGNMENT\_STATE.id ermittelt. In Access haben Aufträge keinen Status hier wird die ID von ‚in Arbeit‘ zurückgegeben.
- `get_term_of_payment($dbConn,$contact_id)`
- `get_discount($dbConn,$contact_id)`
- `get_term_of_payment_discount($dbConn,$contact_id)`
- `get_title_for_assignment_item($dbConn,$assignment_id)`  
Es wird der Titel des übergeordneten AST\_ASSIGNMENT ermittelt.
- `adjust_list_price($dbConn,$list_price,$assignment_id)`  
In der Preisangabe der Tabelle Auftrag.Listenpreis steht der Preis für eine Anzeige innerhalb diese Auftrags. In den Preisangaben von AST\_ASSIGNMENT.list\_price hingegen die Summe aller untergeordneten AST\_ASSIGNMENT\_ITEMS. Die Summe von AST\_ASSIGNMENT wird durch diese Funktion aktualisiert, indem die Preise der AST\_ASSIGNMENT\_ITEMS nach Einfügen oder Änderung eines Datensatzes aufaddiert werden und in AST\_ASSIGNMENT.list\_price geschrieben werden.
- `adjust_net_price($dbConn,$net_price,$assignment_id)`  
gleiches Vorgehen wie bei `adjust_list_price()` nur mit `net_price`
- `get_accounting_date_pp($rechnungsdatum)`
- `get_vat_id($p_dbConn,$ust)`
- `get_accounting_date($dbConn,$assignment_id)`
- `get_settlement_date_salesman($provstatus)`

- `insert_edition($dbConn,$ausgabe,$region)`  
Es wird geprüft, ob ein Eintrag in PUB\_EDITION zu dieser Region mit dieser Ausgabe existiert. Falls nicht wird diese PUB\_EDITION neu angelegt und zurückgegeben. Es wird keine neue PUB\_PUBLICATION angelegt!
- `get_placement($placierung)`
- `get_placement_customer_preference($anzeige)`
- `get_flag_colour($farbe)`  
In der Access Datenbank sind die Anzahl der Farben gespeichert, wohingegen in der Postgre Datenbank nur ein Flag gesetzt wird, ob die Anzeige farbig ist oder nicht. Eine Farbe bedeutet, dass `flag_colour` ‚false‘ ist. Zwei oder mehr Farben heißt `flag_colour` ist ‚true‘.
- `get_format_id($a_dbConn,$p_dbConn,$format)`  
Es wird geprüft, ob das Format schon in der Tabelle AST\_AD\_FORMAT existiert, falls nicht, wird es neu angelegt und die ID zurückgegeben
- `get_measure_id($dbConn,$einheit)`

### Die Klasse ‚A2P\_filler\_contact‘

Methoden für INSERT und UPDATE Statements der CONTACT Daten in Richtung Postgre.

- `get_name1($firma)`  
SYS\_CONTACT.name1 ist ein Pflichtfeld. Sollte der String \$firma leer sein wird ein ‚?‘ eingefügt.
- `get_name2($firma2)`
- `get_telephone_for_contact($telephone)`
- `get_fax($fax)`
- `get_email($email)`
- `get_internet($internet)`
- `get_top($top)`
- `get_topd($topd)`
- `get_discount($discount)`
- `get_flag_active($tot)`  
\$tot == ‚true‘ bedeutet, dass `flag_active` ‚false‘ sein muss und andersrum.
- `get_street($strasse)`
- `get_zip_code($plz)`
- `get_city($ort)`
- `get_bank_name($bank)`  
SYS\_BANK\_ACCOUNT.bank\_name ist ein Pflichtfeld. Sollte \$bank ein leerer String sein, so wird ‚?‘ eingefügt
- `get_account_holder($ktoinh);`  
wie `get_bank_name()`
- `get_salutation($anrede)`  
wie `get_bank_name()`
- `get_firstname($vorname)`
- `get_lastname($nachname)`  
wie `get_bank_name()`
- `get_telephone_for_contact_user($telefon,$telefon2)`  
Sollte ein Wert in \$telefon und \$telefon2 stehen, so wird \$telefon2 für den SYS\_CONTACT\_USER zurückgegeben, sonst \$telefon
- `get_mobile($mobil)`

- `get_login($vorname,$nachname,$email,$id,$p_dbConn)`  
Erste Option für das Login ist die eMail Adresse. Sollte diese leer sein, wird ‚\$vorname.\$nachname‘, genommen, aber nur, wenn beide Namen nicht leer sein. Ansonsten wird die SYS\_CONTACT.id genommen. Die Funktion `login_exit()` prüft ob das Login schon existiert und ändert es, indem ein Zähler ‚\_X‘ angehängt wird (z.B. Hans.Meiser\_3).
- `login_exists($login,$p_dbConn)`

## Die Klasse ‚P2A\_filler\_assignment‘

Methoden für INSERT und UPDATE Statements der ASSIGNMENT Daten in Richtung Access.

- `trans_db_date($date)`  
Macht aus dem Datumsformat ‚2004-03-15‘, ‚15.03.2004‘
- `get_format($p_dbConn,$assignment_id)`
- `get_farbe($p_dbConn,$assignment_id)`  
Falls `flag_colour = 'true'` ist, wird Farbe auf 4 gesetzt, sonst 1
- `get_listenpreis($p_dbConn,$assignment_id)`
- `get_nettopreis($p_dbConn,$assignment_id)`
- `get_region($p_dbConn,$assignment_id)`
- `get_verkaeufer($p_dbConn,$a_dbConn,$salesman_id)`  
Es wird der Nachname des Verkäufers ermittelt und geprüft ob dieser in der Access Datenbank als Verkäufer eingetragen ist. falls nicht, wird dieser als Verkäufer neu angelegt.
- `get_aprovision($aprovision)`
- `get_adatum($assignment_date)`
- `get_malstaffel($a_dbConn,$assignment_item)`
- `get_placierung($p_dbConn,$assignment_id)`
- `get_bankeinzug($bank_account_id)`  
Sollte ein Wert in AST\_ASSIGNMENT.bank\_account\_id stehen, so wird ‚true‘ zurückgegeben, sonst ‚false‘
- `get_placierung_kundenwunsch($p_dbConn,$assignment_id)`  
Ist der String in AST\_ASSIGNMENT\_ITEM nicht leer wird ‚true‘ zurückgegeben, sonst ‚false‘
- `get_status($p_dbConn,$a_dbConn,$state_id)`
- `get_ausgabe($p_dbConn,$edition_id)`
- `get_status_anzeige($p_dbConn,$a_dbConn,$ai_id)`
- `get_prov_status($settlement_date_salesman)`
- `get_zahlung($p_dbConn,$ai_id)`  
Wird aus AST\_DOCUMENTs des AST\_ASSIGNMENT\_ITEMS ermittelt
- `get_zahlungseingang($p_dbConn,$ai_id)`  
wie `get_zahlung()`
- `get_einheit($p_dbConn,$measure_id)`
- `get_preis_e($list_price)`
- `get_ust($p_dbConn,$measure_id)`
- `get_bezeichnung($title)`
- `get_rechnungsdatum($p_dbConn,$accounting_date)`
- `get_rechnungsnummer($p_dbConn,$ai_id)`  
wie `get_zahlung()`

## Die Klasse ,P2A\_filler\_contact'

Methoden für INSERT und UPDATE Statements der Kontakdaten in Richtung Access.

Bei diesen Funktionen wird meistens die SYS\_CONTACT.id des Kontaktes und eine Liste aller ,SYS\_CONATCT\_USER' in einem Array übergeben. Aus dem Array wird dann der passende default Eintrag gesucht.

- get\_anrede(\$contact\_id,\$array\_contact\_user)
- get\_vorname(\$contact\_id,\$array\_contact\_user)
- get\_nachname(\$contact\_id,\$array\_contact\_user)
- get\_firma(\$name1)
- get\_firma2(\$name2)
- get\_strasse(\$contact\_id,\$array\_address)
- get\_plz(\$contact\_id,\$array\_address)
- get\_ort(\$contact\_id,\$array\_address)
- get\_telefon(\$telephone)
- get\_telefax(\$fax)
- get\_mobil(\$contact\_id,\$array\_contact\_user)
- get\_www(\$internet)
- get\_e\_mail(\$email)
- get\_tot(\$flag\_acitve)  
ist \$flag\_active == true, so wird 'false' zurückgegeben, sonst ,true'
- get\_blz\_kunde(\$contact\_id,\$array\_bank\_account)
- get\_kto\_nr\_kunde(\$contact\_id,\$array\_bank\_account)
- get\_ftoinh\_kunde(\$contact\_id,\$array\_bank\_account)
- get\_zahlungsziel(\$top)
- get\_zahlungszielskonto(\$topd)
- get\_skonto(\$discount)

## Die Klasse ,DocumentClass'

Zu jedem AST\_ASSIGNMENT\_ITEM gibt es AST\_DOCUMENTs, wie Rechnung, und Mahnungen.

Diese Klasse erstellt beim neu Anlegen eines AST\_ASSIGNMENT\_ITEMS die dazugehörigen AST\_DOCUMENTs.

Variablen sind:

- \$dbConn  
Beinhaltet die Postgre Datenbankverbindung
- \$assignment\_item\_id
- \$related\_doc\_id  
sollten mehr als ein Dokument erstellt werden, steht hier die ID des Bezugsdokuments
- \$type  
gibt an welches Dokument erstellt wird
- \$state
- \$address
- \$debit\_value
- \$actual\_value

- \$invoice\_number

Methoden:

- set\_address(\$contact\_id)  
Es wird die Adresse aus name1, salutation, firstname, lastname, street, zip\_code, city zusammengesetzt, die dann beim Anlegen der Dokumente in die Datenbank eingetragen werden.
- createDocs()  
steuert die Erstellung der AST\_DOCUMENTs bezüglich des AST\_ASSIGNMENT\_ITEMS je nach gesetztem Typ. Es werden alle Dokumente, die vor dem eigentlich zu Erstellenden Dokument angelegt. (Bsp.: \$type ist Zweite Mahnung: Es wird die Rechnung, dann die erste und dann die zweite Mahnung erstellt)
- createAccount()  
Es wird die Rechnung erstellt.
- createFirstDemand()  
es wird die erste Mahnung erstellt.
- createSecondDemand  
Es wird die zweite Mahnung erstellt.
- createLastDemand()  
Es wird die letzte Mahnung erstellt.
- get\_related\_doc\_id(\$last\_oid)  
die Variable \$related\_doc\_id wird gesetzt
- doc\_ass\_item()  
Die Zuordnung des AST\_ASSIGNMENT\_ITEM zum erstellten AST\_DOCUMENT wird in der Tabelle AST\_DOCUMENT2AST\_ASSIGNMENT\_ITEM gesichert.

### 3.2.4 Einschränkungen während parallelen Betriebs

Einige Eigenschaften werden in den Datenbanken unterschiedlich gehandhabt, deshalb müssen in der Synchronisationsphase einige Einschränkungen eingehalten werden.

Eigenschaften (wie Preise, Format,...) von Anzeigen werden in dem zugehörigem Auftrag verwaltet. Diese Eigenschaften sind für alle zugeordneten Anzeigen gültig.

In der neuen Datenbank werden diese Werte in den AST\_ASSIGNMENT\_ITEMS verwaltet und können sich von anderen AST\_ASSIGNMENT\_ITEMS unterscheiden, obwohl diese zum selben AST\_ASSIGNMENT gehören.

Damit problemlos synchronisiert werden kann, sollte dies beim Anlegen neuer AST\_ASSIGNMENTS berücksichtigt werden.

Folgende Felder der AST\_ASSIGNMENT\_ITEMS eines AST\_ASSIGNMENTS sollten immer die gleichen Werte enthalten:

- AST\_ASSIGNMENT\_ITEM.list\_price, AST\_ASSIGNMENT\_ITEM.net\_price  
Diese Preise sollten für alle AST\_ASSIGNMENT\_ITEMS eines AST\_ASSIGNMENT gleich sein. In der Access Datenbank bestimmt der Preis im Feld Auftrag.Listenpreis bzw. Auftrag.Nettopreis den Preis aller dazugehörigen Anzeigen. Der Wert im Feld AST\_ASSIGNMENT.list\_price bzw. AST\_ASSIGNMENT.net\_price ist die Summe aller AST\_ASSIGNMENT\_ITEM.list\_price/net\_price.

- AST\_ASSIGNMENT\_ITEM.ast\_ad\_format\_id  
Das Format wird in der Access Datenbank ebenfalls im Auftrag für alle dazugehörigen Anzeigen verwaltet, ganz im Gegensatz zum neuen Datenbank Design, wo jeder AST\_ASSIGNMENT\_ITEM ein andere Format haben kann.
- AST\_ASSIGNMENT\_ITEM.placement,  
AST\_ASSIGNMENT\_ITEM.placement\_customer\_preference  
Diese Felder sollten unter einem AST\_ASSIGNMENT auch gleich sein.
- AST\_ASSIGNMENT\_ITEM.flag\_colour  
Sollten unter einem AST\_ASSIGNMENT auch immer gleich sein
- AST\_ASSIGNMENT\_ITEM.pub\_publication\_id  
Hier sollten die Editionen aller ASSIGNMENT\_ITEMS eines AST\_ASSIGNMENT die gleiche PUB\_PUBLICATION.id haben. In der Access Datenbank wird nämlich die Region (Dies entspricht ja der PUB\_PUBLICATION) in Auftrag.Region verwaltet. Die Ausgabe steht in der jeweiligen Anzeige.

### 3.3 Probleme bei der Synchronisation

Es war teilweise nicht trivial die Daten problemlos von der einen Datenbank in die andere zu übernehmen. Das lag einmal an den unterschiedlichen Datenbankstrukturen.

Folgende Probleme traten auf:

- neue ‚NOT NULL‘ Felder im neuen Datenbankdesign, für die es in Access keine Daten gab
- unterschiedliche Datentypen in den zusammengehörenden Feldern
- Felder an anderer Stelle im Design (Auftrag.Format – AST\_ASSIGNMENT\_ITEM.ad\_format\_id),
- unterschiedliche Datumsformat Interpretation (Postgre interpretierte das Datum 02.03.2002 als 3 Februar 2002 aber das Datum 14.03.2002 als 14.März 2002)
- fehlende Einträge. Es mussten Dummydaten eingetragen werden, die bei der Rückrichtung ignoriert werden mussten (z.B. AST\_CONTACT\_USER.lastname = '?')
- doppelte Einträge (in der Access Datenbank gab es teilweise doppelte Stammdaten Einträge mit unterschiedlichen Werten, die bei der Ermittlung der benötigten Daten für die Übernahme eines Datensatzes Probleme gaben)
- falsche Einträge (In der Access Datenbank waren ungültige Einträge (z.B. Datum) siehe unten.
- Ungültige Aufträge, Projekte, Anzeigen, Projektposten, bei denen wichtige Daten fehlten

### 3.4 Fazit/Bewertung

Wie gut kann Access auf andere Datenbanken abgebildet werden? Diese Arbeit sollte auf diese Frage Antwort geben.

Ein Problem bei einem Datenbankwechsel ist, die Daten in die neue Datenbank zu bringen, und zwar so, dass möglichst alle Daten vollständig erhalten bleiben. Ein gut durchdachtes Mapping zwischen den Datenbanken ist äußerst wichtig. Auch das Format der Daten muss berücksichtigt und teilweise angepasst werden. Gutes Beispiel ist hier das Datumsformat, das bei vielen Datenbanken unterschiedlich aussieht. Es kommt natürlich auch vor, das benötigte Daten im alten System fehlen oder falsch sind. Diese müssten ermittelt und noch korrigiert werden, sonst könnte es während der Datenübernahme zu Fehlern kommen. Nur der Testlauf mit allen Daten der aktuellen Datenbank kann sicher stellen, ob alle Daten richtig verarbeitet werden,

da es erfahrungsgemäß sehr schwer ist, *alle* Möglichkeiten zu bedenken und zu testen.

Sollte sich das Design der Datenbanken sehr unterscheiden, ist das Mapping natürlich schwieriger und es müssen oft Kompromisse gefunden werden. Beispielsweise könnten Daten im neuen Design verlangt werden, die beim Alten gar nicht vorhanden sind. In diesem SEP mussten beispielsweise zusätzliche Methoden geschrieben werden, die die benötigte Daten aus unterschiedlichen Tabellen zusammensuchen. Sollten die Daten gar nicht ermittelt werden können, müssen Dummydaten eingetragen werden.

Das Übernehmen der Daten von eine in die andere Datenbank ist der eine Schritt. Etwas komplizierter wird es, wenn beide Datenbanken immer die selben Daten enthalten sollen, da, wie in diesem Fall, beide Systeme eine Zeit lang parallel laufen und auf beiden Seiten gearbeitet wird.

Es muss hier auch noch die Rückrichtung durchdacht und implementiert werden. Ein Großteil des Vorgehens der Hinrichtung kann für die Rückrichtung relativ einfach umgedreht werden, aber eben nicht alles. Zu beachten sind z.B. Dummydaten, falls beispielsweise Daten im neuen Design benötigt werden, die im alten nicht vorhanden waren und deshalb eingetragen wurden.

Durch die Unterschiede im Design kann es auch sein, dass beim Anlegen von neuen Daten bestimmte Regeln und Einschränkungen eingehalten werden müssen, um garantieren zu können, dass die Daten auf beiden Seiten gleich sind und nicht zu Inkonsistenzen führen. Als Beispiel sei auf das Format Feld der Aufträge verwiesen, das im neuen Design in der untergeordneten Tabelle AST\_ASSIGNMENT\_ITEM zu finden ist. Die Benutzer müssen in diesem Fall darauf achten, dass alle AST\_ASSIGNMENT\_ITEMS eines AST\_ASSIGNMENTS immer das gleiche Format haben, ansonsten können beim Synchronisieren Daten überschrieben werden.

Der Aufwand für ein solches Übernahme- oder Synchronisationstool, ist von einigen Faktoren abhängig und kann sehr variieren. Der Unterschied der Designs spielt eine große Rolle. In dem betrachteten konkreten Fall lagen die Schwierigkeiten dabei, dass Informationen an unterschiedlichen Stellen in den Strukturen gespeichert wurden oder Felder im alten System gar nicht vorhanden waren.

Ein Punkt der ein solches Tool noch komplizierter macht, ist die Realisierung der Rückrichtung.

Bei der Ersetzung eines alten Systems durch ein Neues, ist in den meisten Fällen eine Übernahme der alten Daten erwünscht oder sogar notwendig. Die Frage nach dem ‚wie‘ endet meist in der Entwicklung und Implementierung eines Tools, das die Daten automatisch von der alten Datenbank in die neue übernimmt. Allgemein einsatzfähige Tools sind schwer zu finden, da es sich meistens um sehr spezifische Datenbankeninhalte handelt und im neuen System meist besser angepasste und effizientere Datenbankstrukturen verwendet werden.

Der Aufwand ein solches Tool zu entwickeln ist von einigen Faktoren abhängig, die an diesem konkreten Fall beschrieben wurden.

Sinnvoll und lohnenswert ist der Aufwand allemal, da einmal eine manuelle Übernahme mühselig und meistens wegen den großen Datenmengen nicht machbar ist. Andererseits werden oftmals inkonsistente Daten gefunden, die korrigiert werden können und auch ‚Datenmüll‘ kann letztendlich aussortiert werden.

Um den Aufwand aber möglichst gering zu halten, sollte man nur eine Richtung realisieren, nämlich eine Übernahme der Daten vom alten System in das neue. Die Rückrichtung ist oft nicht so trivial, d.h. je nach Datenbankstrukturen müssen zusätzliche Fälle berücksichtigt werden. Der Datenfluss kann also nicht einfach umgekehrt werden. Auch kann es sein dass während des parallelen Betriebs der beiden Systeme bzw. Datenbanken eine Liste von Einschränkungen bei der Benutzung des neuen oder auch alten Systems eingehalten werden müssen, um die Daten sauber zu halten.

## 4 Zusammenfassung

Die Fallstudie für das Systementwicklungsprojekt (SEP) wurde von gogol medien GmbH eingebracht. Es wurde auf Basis des Turbine-Frameworks aufgebaut. Dieses Projekt ist eine Migration von Access Anwendungen zu auf PostgreSQL Datenbank basierte Web-Anwendungen. Es wurde anfangs mit der MS Access gearbeitet. Wegen der Kosten-, Sicherheit- und Effizienz-Gründe benützen wir jetzt PostgreSQL Datenbank statt MS Access Datenbank. In der Übergangsphase werden die Daten in beiden Datenbanken (Access und PostgreSQL) gesichert. Dazu müssen die Änderungen, die auf einer Seite gemacht werden, auf die andere Datenbank überspielt werden. Diese Aufgabe übernimmt das Synchronisationstool.

Es gibt 2 Use Cases:

1. Synchronisation MS Access → PostgreSQL
2. Synchronisation PostgreSQL → MS Access

Die Synchronisation wird in regelmäßigen Abständen stattfinden. In welche Richtung dies geschieht, wird in der Konfigurationsdatei festgelegt. Die Synchronisation kann auch manuell über das Internet gestartet werden.

Außerdem die Daten in den Tabellen sollen die beigelegte Access Anwendungen und Funktionen (Z.B Anzeigen, Report und Chart) in passende Web-Anwendungen umgestellt werden. In der Realisierung von Anzeigen, Report und Chart wurde das Pull MVC-Model in Turbine streng eingesetzt. Dieses Modell ermöglicht eine sehr schnelle und einfache Änderbarkeit des Ablaufs einer Anwendung und führt zu einer noch größeren Unabhängigkeit zwischen Web-Designer und Java-Entwickler. Diese Architektur ist sehr einfach zu verstehen und zu implementieren.

Im Vergleich mit Access - Visual Basic Anwendungen haben Turbine – PostgreSQL Anwendungen viele Vorteile. Turbine ist ein auf Java Servlets basiertes Framework, das wie Servlets es erlaubt, die Funktionalitäten des Servers zu erweitern. Durch diese Erweiterungen ist es möglich, dass der Client Einfluss auf das erwartete und vom Server gegebene Ergebnis nimmt. Dadurch, dass sie in Java geschrieben werden, können sie programmiertechnisch als gewöhnliche Java-Klassen angesehen werden, auch wenn der Umgang mit den speziellen Methoden zunächst unbekannt erscheint. Die Anwendung von Servlets ist aus vielen Gründen vorteilhaft. So spielt die Tatsache der Unabhängigkeit von Servlets eine große Rolle. Servlets sind zum einen plattform- und serverübergreifend, zum anderen aber auch protokollunabhängig (SMTP, POP3, HTTP...). Die Plattformunabhängigkeit erreichen Servlets dadurch, dass sie in Java geschrieben sind. So können sie ohne Modifikationen auf verschiedenen Plattformen (wie zum Beispiel Unix, Windows,...) verwendet werden. Die Anwendungen von Turbine sind leicht wieder verwendbar und auf andere Framework und Plattform zu portieren. Aber MS Access kann nur auf MS Windows arbeiten.

Turbine garantiert weitestgehend Sicherheit. "Weitestgehend" bedeutet, dass natürlich keine absolute Sicherheit garantiert werden kann. Sicherheit beinhaltet zum einen die Sicherheit vor einem Absturz durch Einsatz einer Java Virtual Machine (JVM). Diese führt das Servlets aus und stellt sicher, dass Servlets keinen Crash durch unerlaubten Zugang zum Datenspeicher verursachen können, weil die JVM keinen Zugang zu diesem erlaubt. Außerdem ruft die JVM eine

Ausnahmebehandlung auf, wenn Fehler entdeckt worden sind, anstatt einen Crash zuzulassen und verifiziert, dass kompilierte Java-Klassen keine illegalen Operationen aufrufen können. Ein anderer Vorteil der Benutzung einer solchen JVM ist, dass Servlets bei der Kompilierung in Java-Bytecode überführt werden. Der Bytecode ist quasi Maschinencode für die JVM. Durch den Vorgang des Kompilierens können Syntaxfehler erkannt und die benutzten Typen auf ihre Richtigkeit überprüft werden. So wird Turbine stabiler und sicherer. Die Kompilierung in Bytecode hat den Vorteil, dass die Servlets schneller in der Ausführung werden. Zum anderen wird Sicherheit vor ungewolltem Datenzugang gewährleistet, einerseits durch den Gebrauch des "Security Managers" des Servers, andererseits durch die Möglichkeit der Authentifizierung und Autorisierung der Benutzer. Die Turbine hat bei jeder weiteren Anfrage Kenntnis der Identität des anfragenden Clients. Deswegen hat Turbine Vorteile bei der Sicherheit und der Stabilität im Vergleich mit MS Access.

Turbine Anwendungen sind Web Anwendungen. Deswegen können mehrere Benutzer in beliebigem Ort gleichzeitig durch Internet dieses System verwenden. Aber MS Access ist nur eine Desktop Anwendung und auch nicht mehrbenutzerfähig.

Vom Kosten hier hat Turbine größere Vorteile als MS Access bei der Lizenzgebühr. Denn Turbine und PostgreSQL sind beide Open-Source Tools. Andererseits soll man die Kosten von Umstellen und Synchronisation der Turbine-Anwendung berücksichtigen. Die Kosten und Aufwand sind abhängig mit der Anzahl von Tabellen, der Komplexität und Abhängigkeit zwischen Tabellen und der Anzahl der angezeigten Tabellen, Report und Chart. Außerdem darf der Aufwand von der Einarbeitung von Turbine und PostgreSQL auch nicht übersehen werden.

Obwohl Turbine und PostgreSQL viele Vorteile haben, haben sie auch Nachteile. MS Access ist sehr benutzerfreundlich und leicht zu benutzen. Z.B. mit MS Access kann man leicht Report und graphische Abfrage ohne besondere Programmierung erstellen. Im Access kann man auch mit Hilfe von anderer MS Tools (z.B. Word, Excel und Frontpage etc.) leicht Diagram und Web Publikation. Aber mit Turbine und PostgreSQL sind die nur für Fachleute möglich. Denn alles muss mit Code realisiert werden. Außerdem sind Turbine und PostgreSQL neue entwickelte Open-Source Projekte. Sie haben nicht so viele und ausführliche Dokumentationen wie MS Access. Es steht nur wenige Hilfsmittel zur Verfügung. Trotzdem ist die Mailingliste sehr hilfreich.

Wir haben viele Open-Source Tools in diesem Projekt verwendet. Durch den Einsatz der Tools Turbine, Torque und Avalon konnten viele Funktionalitäten sehr einfach und vor allem sehr schnell implementiert werden. Für die komponenten-orientierte Konfiguration einer Web-Anwendung war Turbine, mit den sehr vielen Einstellungsmöglichkeiten und der damit verbunden Flexibilität, sehr gut geeignet.

Turbine unterstützt Velocity und JSP als Templatesprache. Die zugrunde liegende Velocity-Template-Language (VTL) besitzt im Vergleich zu anderen Programmiersprachen (Java, VB, etc.), die in Template-Techniken wie JSP oder ASP Verwendung finden, nur wenige Sprachkonstrukte. Dies soll vermeiden, dass Templates die komplette Geschäftslogik enthalten. Deswegen kann die Trennung von Präsentations- und Anwendungslogik besser garantiert werden. In diesem

Projekt wurde strenger MVC Model eingesetzt. Um Präsentation und Anwendungslogik besser zu trennen, wählten wir hier Velocity als Templatesprache.

Durch Torque wurden die teilweise sehr umständlichen Zugriffe auf Datensätze einer Datenbank, wie sie z.B. durch die direkte Verwendung der *JDBC-API* notwendig wären, eingespart. Man konnte sich vom Anfang an auf die Kern-Funktionalitäten konzentrieren. Dagegen hat Torque Probleme bei der Erzeugung von Peerklassen. Um Peerklassen zu erzeugen, muss die entsprechende XML-Schema Datei zuerst von einer SQL-Datei oder einer Datenbank erzeugt werden. Die von der aktuellen Version von Torque angebotene Konvertierungsmethode „sql2xml“ und die durch JDBC konvertierte Methode unterstützen nicht alle Datenbanken (auch PostgreSQL). Wir müssen manche Syntaxfehler in die von der SQL-Datei oder Datenbank konvertierte XML-Datei manuell anpassen.

### **Bedanken:**

Zum Abschluss möchten wir uns bei dem Betreuer des Projekts, Herrn Stefan Wagner, und Herrn Martin Huber bedanken, der uns mit Rat und Tat zur Seite stand. Wir bedanken uns auch herzlich bei Herrn Gunter Miessbrandt und dem kompletten Team der gogol medien GmbH & Co. KG, die uns in jeglicher Hinsicht unterstützten und durch die dieses Systementwicklungsprojekt erst möglich wurde.

# Abbildungsverzeichnis

|  |    |
|--|----|
| Abbildung 2-1 Struktur von dem Turbine Framework .....                   | 5  |
| Abbildung 2-2 Die Module von Turbine.....                                | 6  |
| Abbildung 2-3 Die Modulverschachtelung von Turbine .....                 | 7  |
| Abbildung 2-4 Die Modul-Loader von Turbine .....                         | 8  |
| Abbildung 2-5 Das Pull MVC-Model in Turbine .....                        | 9  |
| Abbildung 2-6 Gesamter Überblick.....                                    | 15 |
| Abbildung 2-7 Struktur der Komponente bad.....                           | 17 |
| Abbildung 2-8 Struktur der Komponente badGUI .....                       | 19 |
| Abbildung 2-9 Anwendungssicht bad und badgui.....                        | 20 |
| Abbildung 2-10 Struktur der Komponente oma.....                          | 21 |
| Abbildung 2-11 Struktur der Komponente omaGUI .....                      | 23 |
| Abbildung 2-12 Anwendungssicht oma und omagui.....                       | 24 |
| Abbildung 2-13 Grobe Struktur von Komponente Chart .....                 | 25 |
| Abbildung 2-14 Klassen-Struktur von Komponente Chart. ....               | 27 |
| Abbildung 2-15 Grobe Struktur von Report.....                            | 28 |
| Abbildung 2-16 Verfeinerte Struktur von Report .....                     | 29 |
| Abbildung 2-17 Klassen-Struktur von Komponente Report. ....              | 31 |
| Abbildung 2-18 Ansicht der Komponente bad .....                          | 35 |
| Abbildung 2-19 Formular zum Anlegen eines neuen Ansprechpartners.....    | 38 |
| Abbildung 2-20 Ablauf addUser in der Komponente bad.....                 | 39 |
| Abbildung 2-21 Formular zum Editieren eines existierenden Eintrags ..... | 40 |
| Abbildung 2-22 Anzeige von Einträgen .....                               | 41 |
| Abbildung 2-23 Ablauf von der Komponente Chart .....                     | 42 |
| Abbildung 2-24 Das Tortendiagramm von Chart .....                        | 43 |
| Abbildung 2-25 Ablauf von der Komponente Report .....                    | 44 |
| Abbildung 2-26 der generierte Bericht in HTML.....                       | 45 |
| Abbildung 3-1 Klassenübersicht Access, Postgre .....                     | 50 |
| Abbildung 3-2 Klassenübersicht Access.....                               | 52 |
| Abbildung 3-3 Klassenübersicht PostgreSQL.....                           | 53 |
| Abbildung 3-4 Klassenübersicht Synchronisation.....                      | 54 |
| Abbildung 3-5 Klassenübersicht Synchronisation.....                      | 54 |
| Abbildung 3-6 Klassenübersicht gesamt .....                              | 55 |
| Abbildung 3-7 Sequenzdiagramm Synchronisation der Kontakte.....          | 57 |
| Abbildung 3-8 Sequenzdiagramm Synchronisation der Auftragsdaten.....     | 58 |

# Literaturverzeichnis

- [1] Orfali R., Harkey D. (1996): *Client/Server Programming with Java and CORBA*, John Wiley & Sons
- [2] Sun Microsystems, Inc (22.02.2004): *Java 2 Platform Standard Edition v. 1.4.2*, <http://java.sun.com/j2se/1.4.2/index.html>
- [3] Sun Microsystems, Inc (22.02.2004): *Java 2 Platform Standard Edition v. 1.4.2, Summary of new features and enhancements*, <http://java.sun.com/j2se/1.4.2/docs/api/>
- [4] J. Hunter, W. Crawford (1998): *Java Servlet Programming*, O'Reilly, Sebastopol
- [5] Apache Software Foundation (20.01.2004): *The Jakarta Site - Apache Tomcat*, <http://jakarta.apache.org/tomcat/index.html>
- [6] Apache Software Foundation (20.01.2004): *The Jakarta Site - Jakarta Tomcat*, <http://jakarta.apache.org/turbine>
- [7] Apache Software Foundation (20.01.2004): *Turbine Features*, <http://jakarta.apache.org/turbine/turbine-2/features.html>
- [8] Apache Software Foundation (20.01.2004): *Turbine Specification*, <http://jakarta.apache.org/turbine/turbine-2.3/fsd.html>
- [9] Apache Software Foundation (20.01.2004): *Turbine Model 2+1*, <http://jakarta.apache.org/turbine/turbine-2/model2+1.html>
- [10] Sun Microsystems, Inc (20.01.2004): *J2EE Design Patterns - Model-View-Controller Architecture*, [http://java.sun.com/blueprints/patterns/j2ee\\_patterns/model\\_view\\_controller/](http://java.sun.com/blueprints/patterns/j2ee_patterns/model_view_controller/)
- [11] D. Kracht (20.01.2004): *Model-View-Controller für Web-Anwendungen*, <http://home.t-online.de/home/dietrich.kracht/mvc2.htm>
- [12] Apache Software Foundation (20.01.2004): *Turbine Pull vs. Push*, <http://jakarta.apache.org/turbine/turbine-2/pullmodel.html>
- [13] Apache Software Foundation (20.01.2004): *Torque*, <http://jakarta.apache.org/turbine/torque>
- [14] Apache Software Foundation (20.01.2004): *Torque Tutorial*, <http://jakarta.apache.org/turbine/torque/tutorial.html>
- [15] Apache Software Foundation (20.01.2004): *Velocity Design*, <http://jakarta.apache.org/velocity/design.html>
- [16] Apache Software Foundation (20.01.2004): *Velocity User Guide*, <http://jakarta.apache.org/velocity/user-guide.html>

[17] Apache Software Foundation (20.01.2004): *Velocity Developer's Guide*,  
<http://jakarta.apache.org/velocity/developer-guide.html>

[18] Apache Software Foundation (20.01.2004): *Apache Ant*,  
<http://jakarta.apache.org/ant/>

[19] CS Component Studio GmbH [www.cai.de/leistungsprofil/komponentendef.htm](http://www.cai.de/leistungsprofil/komponentendef.htm)

[20] Apache Software Foundation (20.01.2004): *Avalon Component Service*  
<http://jakarta.apache.org/turbine/turbine-2.3/services/avalon-component-service.html>