HOCHSCHULE DER MEDIEN

BAMS

Bestands & Auftrags Management System

Softwareprojekt Sommersemester 2005
FB1- Studiengang Medieninformatik

Entwicklerteam:

Sandra Banic, Medieninformatik 7.Semester

Sabrina Ferwagner, Medieninformatik 8.Semester

Christine Meisner, Medieninformatik 7.Semester

Betreuer:

Prof.Walter Kriha

1.Inhalt

2. Einleitung

2.1.Allgemein

BAMS steht für Bestands, Auftrags und Management System.
Dieses System wurde für die Firma „Weiberg“ in Ebersbach umgesetzt.
Aufgrund deren Bedarf einer zentralen Verwaltung der Kunden, des Materialbestands, der Aufträgen und der Lieferanten in elektronischer Form, sowie der automatischen Auftragsabwicklung gab es verschiedene Möglichkeiten der Realisierung.
Eine für das Projekt völlig ausreichende Lösung wäre die Umsetzung der Anforderungen mit `php` oder die Anpassung eines bestehenden Opensource Verwaltungssystems.
Unser Team hatte jedoch die Intension sich mit der `Enterprise-Technologie` auseinanderzusetzen und sich darin einzuarbeiten.
Somit fiel die Entscheidung auf eine J2EE- basierte Lösung.

Bislang verwaltete die Firma „Weiberg“ Ihre Kunden- und Lieferantendaten sowie die Lager- und Bestandsdaten zum Teil in Form eines Karteikartensystems.

2.2.Anforderungen

Anforderungen der Firma „Weiberg“ an das System:

- Kunden- und Lieferantendaten verwalten können
- Lager- und Bestandsdaten verwalten können
- Übersicht über alle laufenden und abgeschlossenen Geschäftsabwicklungen
- Verbesserung der Nutzerfreundlichkeit
- exakte Anpassung an die Bedürfnisse der Firma Weiberg-Technische Produkte
- Verbesserung des Datenmanagements
- daraus resultierend eine Zeitersparnis und Budgeteinsparung

Nach der Auftragserstellung und Bestandskontrolle kann eine personalisierte Rechnung erstellt werden.

2.3.Umsetzung

Ein Teil der Kundendaten wurden in Form von Tabellen verwaltet. Diese Daten werden automatisch ausgelesen und in die Datenbank des BAMS Systems eingegeben. Dies wird einmalig gemacht.
Die Verwaltungsmöglichkeiten des BAMS:

•Kundenverwaltung :
- Kunde anlegen
- Kunde ändern
- Kundensuche anhand Name, Kundennummer, Firma
    - alle Kunden oder einzelne Kunden

•Lieferantenverwaltung :
- Lieferanten anlegen
- Lieferanten ändern

- Lieferantensuche anhand Name, Lieferantennummer, Firma
    - alle Lieferanten oder einzelne Lieferanten

•Artikelverwaltung :
- Artikel anlegen
- Artikel ändern
- Artikelsuche anhand Bezeichnung, Artikelnummer, Matchcode
    - alle Artikel oder einzelne Artikel

•Auftragsverwaltung :
- Auftrag anlegen
- Auftrag ändern
- Generierung der Rechnung für einen Auftrag
- All oder einzelne Aufträge anzeigen

•Lagerverwaltung :
- wird noch implementiert

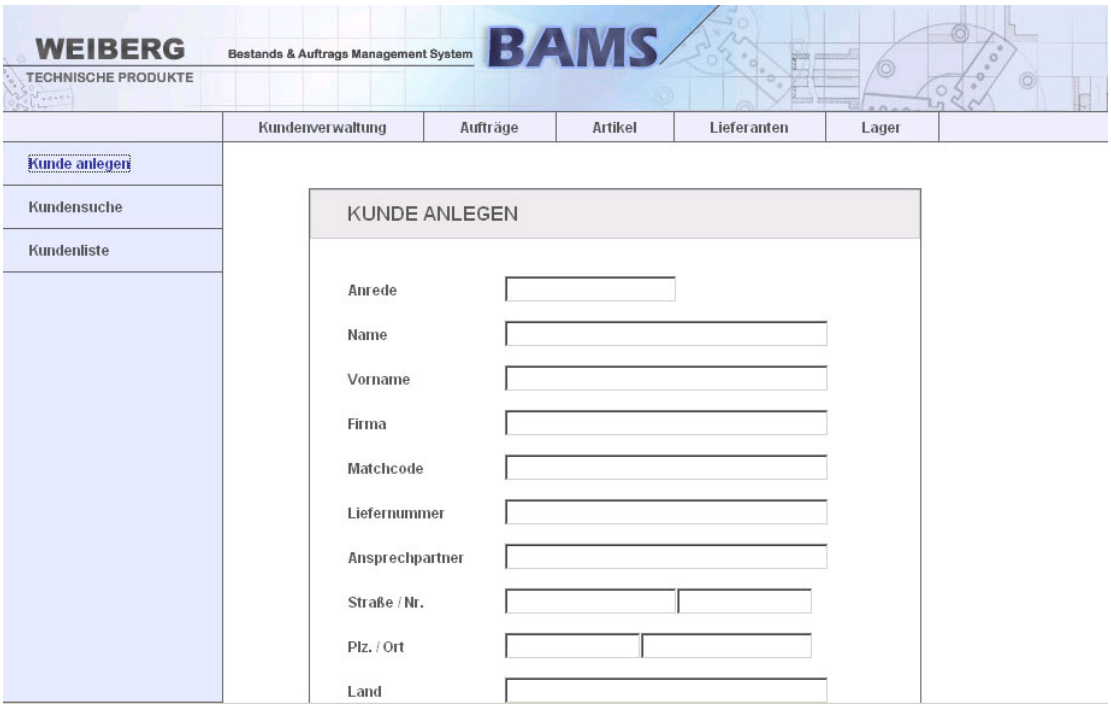


Abbildung 1: Ansicht der Kundenverwaltung

3. Voraussetzungen/Anforderungen

3.1. Verwendete Technologien

Folgende Technologien werden für die Realisierung dieses Projektes benötigt:

- Applikationsserver : jboss 4.0.2 RC1
- Java : j2sdk1.4.2_08
- Datenbank : mysql 4.1.8
- Webcontainer : jakarta-tomcat-5.5.4

Im weiteren Verlauf der Dokumentation werden die Versionsnummern der verschiedenen Technologien nicht benannt es ist von den oben beschriebenen auszugehen.

3.1. Funktion

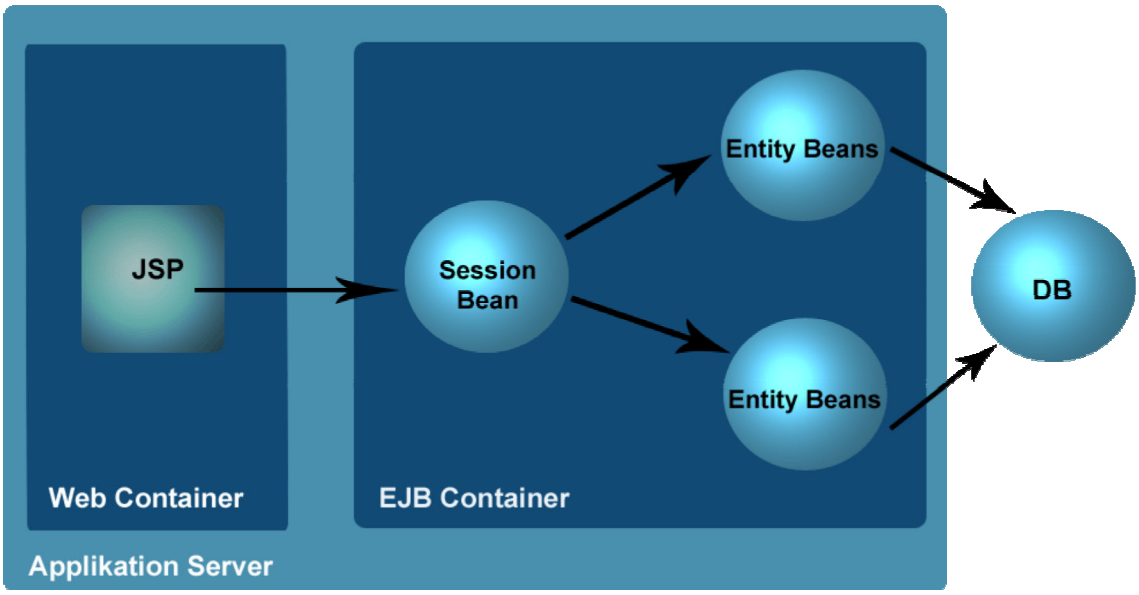


Abbildung 2: Zusammenspiel einzelner Technologien

Dieses Programm ist eine webbasierte Applikation, die über die Komponenten JSP und Servlets, Enterprise Java Beans und der Datenbank MySQL implementiert wird.

4.Aufbau

4.1Web-Tier

Aufbau

Das Web Tier der Anwendung besteht aus JSPs und Servlets.
Die Elemente des Web-Tier stellen die direkte Schnittstelle zum Client-Tier dar, d.h.
sie nehmen Anfragen des Client entgegen und interpretieren Benutzereingaben, des Weiteren bereiten sie die Ergebnisse in HTML oder XML auf und senden diese an den Client zurück.
Über HTTP- Request und HTTP-Response werden die Clientanfragen be- und verarbeitet.
Ein HttpServlet erhält requests von einem Webserver und schickt responses zurück. Die wichtigsten Methoden dieser Klasse sind:

- doGet(HttpServletRequest, HttpServletResponse): Verarbeiten von GET requests.
- doPost(HttpServletRequest, HttpServletResponse): Verarbeiten von POST requests (also requests, die Daten an das Servlet senden).
- init(ServletConfig): Wird einmal aufgerufen, wenn das Servlet vom Server geladen wird. Hier kann man Befehle zur Initialisierung des Servlets platzieren (z.B. Daten aus einer Datenbank holen, die sich während der Laufzeit nicht ändern).
- destroy(): Wird einmal aufgerufen, wenn das Servlet vom Server aus dem Speicher entfernt wird.

Java Server Pages

JavaServer Pages, abgekürzt JSP, ist eine von Sun Microsystems entwickelte Technologie, welche im Wesentlichen zur einfachen dynamischen Erzeugung von HTML- und XML- Ausgaben eines Webservers dient.
Sie erlaubt es Java-Code und spezielle JSP- Aktionen in statischen Inhalt einzubetten. Dies hat den Vorteil, dass die Logik unabhängig vom Design implementiert werden kann. Über die JSP-Syntax können vordefinierte Funktionalitäten in die Seiten eingebaut werden. Diese Aktionen sind in Tag Libraries als Erweiterung der HTML oder XML Tags definiert.
Unter der Verwendung eines speziellen JSP- Compilers werden die Java Server Pages in Java Quellcode umgewandelt. Mittels der JSP- Seiten werden in der BAMS- Applikation die selbst definierten Servlets angesprochen. Umgekehrt werden dem Servlet die JSP- Seiten der zuständigen Ausgabe zugeordnet.
Die Servlets arbeiten über das DTO- Pattern und spezielle Java-Klassen, die Client Anfragen ab.

Grundsätzlich lassen sich JSP als eine Art HTML- oder XML- Seite mit zusätzlichen JSP-spezifischen Tags und Java-Code beschreiben. Eine JSP kann grob in die folgenden Elemente aufgeteilt werden:

- statischer Inhalt wie HTML
- JSP- Direktiven
- JSP- Skriptelemente
- JSP- Aktionen
- JSP Tag-Bibliotheken (Tag Libraries)

JSP Deklarationen

JSP Deklarationen dienen zur Definition von Methoden und Variablen die von anderen Elementen in der Seite verwendet werden können. Innerhalb der JSP wird durch Deklarationen keine Ausgabe erzeugt.
Deklarationen verwenden folgende Syntax:

```
<%! int klassenVariable = 0; %>
```

Ausdrücke

Um Variablen oder Methoden direkt in den HTML Ausgabestrom schreiben zu können, werden Expressions (Ausdrücke) verwendet.

```
<%= variableMeinerKlasse %>
```

Skriptlets

JSP- Skriptlets können zur Implementierung der Ablauflogik verwendet werden, ebenso wie zur Ausgabe in dem Dokument.

```
<% int variable = 0; out.println("Der Wert der Variable ist: " + variable);
%>
```

Kommentare

Kommentare sind nur innerhalb der JSP sichtbar, sie werden nicht in den Ausgabestrom geschrieben.

```
<%-- Kommentar innerhalb einer JSP --%>
```

Aktionen

Aktionen binden die Funktionalität von Webservern ein.
Mögliche Aktionen sind unter anderem:

```
jsp:param
jsp:plugin
jsp:fallback
jsp:setProperty
jsp:getProperty
jsp:useBean
jsp:include
```

Die Java Server Pages der Anwendung BAMS bestehen zum größten Teil aus reinem HTML- Code. Nur die wichtigsten Befehle befinden sich in den Pages, um die Trennung von Java- und HTML-Code zu gewährleisten und nicht um Quick und Dirty zu programmieren.
Mit dieser Architektur ist es gewährleistet, dass bei einer möglichen Umgestaltung, diese auch von einem reinen HTML-Designer vorgenommen werden kann. Der Code ist aus diesem Grund klar strukturiert und übersichtlich gehalten, auch um den Anforderungen des MVC Pattern nahe zu kommen.
Beim MVC Pattern handelt es sich um ein Entwurfsmuster zur Trennung von Programmeigenschaften.
Es werden drei Einheiten berücksichtigt:
- Modell
- View
- Controller

Das Modell ist für ein flexibles Programmdesign zuständig. Dies ist unabdingbar, um spätere Änderungen so einfach wie möglich zu gestalten. Ein weiterer Schwerpunkt des Modells dient der

Wiederverwendbarkeit von einzelnen Komponenten. Durch das Reduzieren von Komplexität bleibt die Anwendung übersichtlich und überschaubar.
Ein großer Vorteil des MVC Konzepts ist die mögliche Rollenverteilung der verschiedenen Fachkräfte.

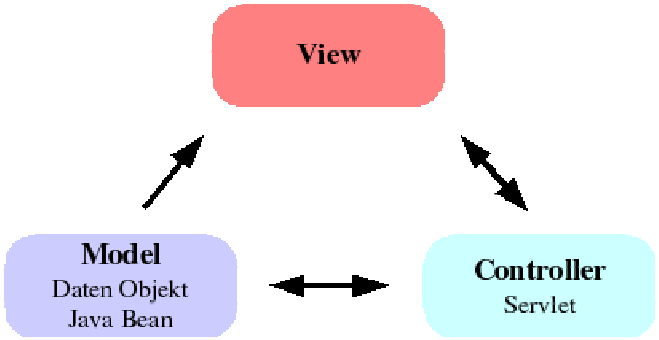


Abbildung 3: Darstellung MVC

Servlets

Im Rahmen der Java 2 Plattform Enterprise Edition (J2EE) werden Java-Objekte als Servlets bezeichnet, die folgende Funktion erfüllen:
Ein Servlet delegiert die Nachfragen des Client an den Webserver und erzeugt die Antwort an den Client. Der Inhalt der Antwort wird zum Zeitpunkt der Anfrage generiert und kann somit den aktuellen Datenbankinhalt wiedergeben.
Servlets sind Instanzen von Java-Klassen, die von der durch die Spezifikation definierten Klasse `javax.servlet.HttpServlet` abgeleitet werden. Die Instanzen werden bei Bedarf über eine Laufzeitumgebung in einem Webcontainer erzeugt. Der Webcontainer kommuniziert mit dem Webserver. Die Servlets stellen im Zusammenhang mit dem MVC-Pattern den Controller dar, während die JSPs den View repräsentieren.
Die Zusammenarbeit der Servlets mit den JSPs, also die Zuordnung bestimmter Aktionen zwischen Servlets und JSPs, wird in einer XML-Datei beschrieben, der sogenannten `web.xml`.
Die Datei `web.xml` dient als Deployment Descriptor der Webanwendung und mappt JSP Seiten und zugehöriges Servlet.

```
<servlet>
<servlet-name>kundenServletAuslese</servlet-name>
<servletclass>
      de.project.kunde.servlet.kundenServletAuslese
</servlet-class>
      </servlet>
<servlet-mapping>
      <servlet-name>kundenServletAuslese</servlet-name>
      <url-pattern>/KundenListe</url-pattern>
</servlet-mapping>
```

Ausschnitt aus der `web.xml` der BAMS-Application

4.2 Middle-Tier

EJB Komponenten

Um das Projekt BAMS zu realisieren sind insgesamt neun EJB Komponenten entwickelt worden. Kunde, Auftrag, Auftrags Fassade, Artikel, Artikel Fassade, Lager, Lager Fassade, Auftragsposition und Lieferant.
Die EJB Komponenten Kunde, Auftrag, Artikel, Lieferant, Lager und Auftragsposition sind als Entity Bean Komponenten umgesetzt, die anderen Komponenten als Stateless Session Beans.

Erklärung der Begriffe Entity und Session Bean

EntityBeans

Entity Beans repräsentieren eine Objekt Sicht auf die persistenten Daten, die in der Datenbank gespeichert sind.
Es gibt bei der Spezifikation EJB 2.1 zwei Ausprägungen von Entity Beans. Entity Beans mit Bean- Managed Persistence (BMP) und Container-Managed Persistence (CMP). In dem Projekt BAMS wurden ausschließlich CMP Entity Beans programmiert und eingesetzt.
Um ein CMP Bean erstellen zu können muss man ein Home Interface, ein Component Interface, die Bean Klasse, eine Primary Key Klasse und den Deployment Descriptor programmieren.

Home Interface

Um eine Entity Bean aufrufen zu können, muss ein Remote Home Interface oder/ und ein Local Home Interface zur Verfügung stehen. Mit diesem Interface erzeugt man neue Bean- Instanzen, löscht bestehende und die Finder Methoden sind dort deklariert, welche im Deployment Descriptor vor zu finden sind. Das Home Interface erbt von EJBHome, wenn es eine Remote Interface, EJBLocalHome, wenn es ein Local Interface ist.
Die Entity Beans werden im Projekt, wenn sie über eine Session Fassade aufgerufen werden, immer über das lokale Interface aufgerufen. Dies bringt einige Performancesteigerungen mit sich, da keine Remote Kommunikation aufgebaut werden muss.

Component Interface

Bei diesem Interface gibt es auch wiederum die zwei Ausprägungen Remote und Local Interface. Das Component Interface kapselt die Zugriffe des Clients auf die Entity Bean. Hier werden die Geschäftsmethoden der Bean deklariert. Zum einen die Getter und Setter- Methode, die den Zugriff auf die Persistenzschicht erlauben. Zum anderen werden hier auch die normalen Geschäftsmethoden deklariert. Das Component Interface erbt von EJBObject oder EJBLocalObject.

Bean Klasse

Alle im Component Interface deklarierten Methoden müssen in der Bean Klasse implementiert werden. Die Bean Klasse muss zwingend neben den Geschäftsmethoden das Interface

javax.EJB.EntityBean implementieren. Dies bedeutet, dass man Callback Methoden über die Bean Zustandsänderungen in ihrem Lebenszyklus implementieren muss. Dazu gehören: EJBPassivate, EJBActivate, EJBLoad, EJBStore, EJBRemove, setEntityContext, unsetEntityContext und die EJBCreate Methode.
Im Projekt wurden aber nur die EJBCreate und setEntityContext Methoden wirklich benötigt und mit „Inhalt" gefüllt. Es war aber sehr interessant zu sehen, wann der EJB Container die jeweiligen anderen Methoden zum Lifecycle der Bean aufruft. Um zu verstehen wie und warum der EJB Container die Methoden aufruft wurden die Methoden testweise mit Ausgaben versehen.
In der Bean Klasse werden auch die EJB Select Methoden deklariert. Dies ist eine spezielle Form der Finder Methoden. Finder Methoden können nur vorhandene fachliche Entitäten zurück liefern, z.B. ein Lager-Objekt. EJB Select Methoden können hingegen beliebige Javatypen zurückliefern. Dadurch wurde im Projekt bei der Generierung einer neuen Entität automatisch eine fortlaufende Nummer vergeben.

Primary Key

Entitäten müssen immer eindeutig durch einen Primarschlüssel identifizierbar sein. Dies wird mit dieser Klasse erreicht. Im Projekt wurden einfache Primary Key Klassen verwendet. Die Schlüsselattribute der Primary Key Klasse muss identisch mit den persistenten Attributen in der Bean Klasse sein.

Deployment Descriptor

Jede EJB Komponente benötigt einen Standard Deployment Descriptor EJB-jar.xml. In diesen Descriptoren werden deklarativ die Konfigurationseinstellung der EJB-Komponenten hinterlegt. Diese werden von dem EJB Container verwendet, um notwendigen Code zu generieren und das Laufzeitverhalten der Bean Komponente zu steuern. Es werden hier die JNDI Namen, über die eine Komponente angesprochen wird, das Persistenzverhalten, das Transaktionsverhalten und die Beziehungen zwischen den EJB Komponeten deklariert.
Neben dem Descriptor EJB-jar.xml gibt es zwei weitere. Der zusätzliche herstellerspezifische Deployment Descriptor jboss.xml und jbosscmp-jdbc.xml, in welchem die Einstellungen für die CMP Beans deklariert sind, damit diese per JDBC auf die Persistenzschicht zugreifen.

Finder Methoden

```
<query>
  <query-method>
       <method-name>findAll</method-name>
       <method-params>
       </method-params>
       </query-method>
       <EJB-ql><![CDATA[Select object(c)from Lager as c]]></EJB-ql>
</query>
```

Hier ist ein Auszug aus dem Deployment Descriptor EJB-jar.xml zu sehen. Der Codeausschnitt zeigt eine Finder Methode. Die Finder Methode ist in der EQL, in Enterprise Java Bean Query Language, programmiert. Diese Methode findAll liefert alle Lagerobjekt aus der Datenbank.

EJB Select

```
<query>
      <query-method>
             <method-name>EJBSelectMaxLager</method-name>
             <method-params/>
       </query-method>
             <EJB-ql>
                  <![CDATA[SELECT MAX(f.lagernummer) FROM Lager f]]>
            </EJB-ql>
</query>
```

Dieser Auszug zeigt eine EJB Select Methode. Diese Methode liefert einen Integer mit der maximalen Lagernummer zurück.

Beziehungen zwischen 2 Komponenten

```
<EJB-relation>
      <EJB-relation-name>Auftragsposition-Artikel</EJB-relation-name>
      <EJB-relationship-role>
        <EJB-relationship-role-name>ein Artikel hat mehrere
Positionen</EJB-relationship-role-name>
        <multiplicity>One</multiplicity>
        <relationship-role-source>
          <EJB-name>Artikel</EJB-name>
         </relationship-role-source>
        <cmr-field>
          <cmr-field-name>auftragsposition</cmr-field-name>
          <cmr-field-type>java.util.Collection</cmr-field-type>
        </cmr-field>
      </EJB-relationship-role>
```

Hier ist ein Auszug aus EJB-jar.xml. Es zeigt einen Teil einer Beziehung zwischen den Beans Auftragsppsition und Artikel.
Diese Beziehung wurde über das Container-Managed Relationship deklariert. Container-Managed Relationship (CMR) bedeutet, dass der EJB-Container verantwortlich für den Zugriff auf die Persistenzschicht ist. Im Deployment Descriptor müssen lediglich die Beziehungen zwischen den Beans deklariert werden.
Es gibt im CMR gibt es drei verschiedene Kardinalitäten. Eine Eins-zu-Eins, eine Eins-zu-Viele und eine Viele-zu-Viele Beziehung. Im Projekt wurden die letzten beiden Beziehungen benötigt.

Eins-zu-Viele Beziehung:

- Kunde und Auftrag → ein Kunde kann mehrere Aufträge haben, aber ein Auftrag gehört nur zu einem Kunden
- Lieferant und Auftrag
- Artikel und Lager

Viele-zu-Viele Beziehung:

- Auftrag und Lager

Session Beans

Es gibt zwei Ausprägungen von Session Beans. Stateful und Stateless Session Bean. Im Projekt wurden ausschließlich Stateful Session Beans. Session Bean sind nicht persistente Komponenten, welche die Logik der Geschäftsprozesse implementieren. Wie bei den Entity Beans benötigt man zu der eigentlichen Bean Klasse, ein Component und Home Interface und den Deployment Descriptor. Stateless Session Beans sind zustandslos und sind nur für die Dauer eines einzelnen Methodenaufrufs den Client zugeordnet.

Home Interface

Das Home Interface einer bietet Methoden zum Erzeugen und Löschen von Session Beans und folgt dem Pattern Factory. Es gibt zwei Ausprägungen Remote und Local Home Interfaces. Für die Beans im Projekt wurden diese über das Remote Home Interface angesprochen.

Component Interface

Das Component Interface repräsentiert die Geschäftsmethoden, welche dem Client zur Verfügung gestellt werden. Es kapselt die Implementierung des Beans. Hierbei gibt es auch wieder die zwei Ausprägung Remote und Local Interface.

Bean Klasse

Die Bean Klasse implementiert die Geschäftslogik der Methoden die im Component Interface deklariert wurden.

Deployment Descriptor

Hier werden wie bei den Entity Beans deklarativ die Konfigurationseinstellung der `EJB-` Komponenten beschrieben. Bei Session Beans benötigt man nur zwei Descriptoren. `EJB-jar.xml` und den herstellerspezifischen Descriptor `jboss.xml`

```
<session >
         <display-name>AuftragsFassade</display-name>
         <EJB-name>AuftragsFassade</EJB-name>
         <home>de.projekt.auftrag.fassade.AuftragsFassadeHome</home>
         <remote>de.projekt.auftrag.fassade.AuftragsFassade</remote>
         <EJB-class>de.projekt.auftrag.fassade.AuftragsFassadeSession</EJB-
class>
         <session-type>Stateless</session-type>
         <transaction-type>Container</transaction-type>
</session>
```

Hier ist ein Auszug der Session Komponente AuftragsFassade. Es ist eine Stateless Session Bean, deren Transaktionsverhalten von Container gesteuert wird.

Design Patterns der Komponenten

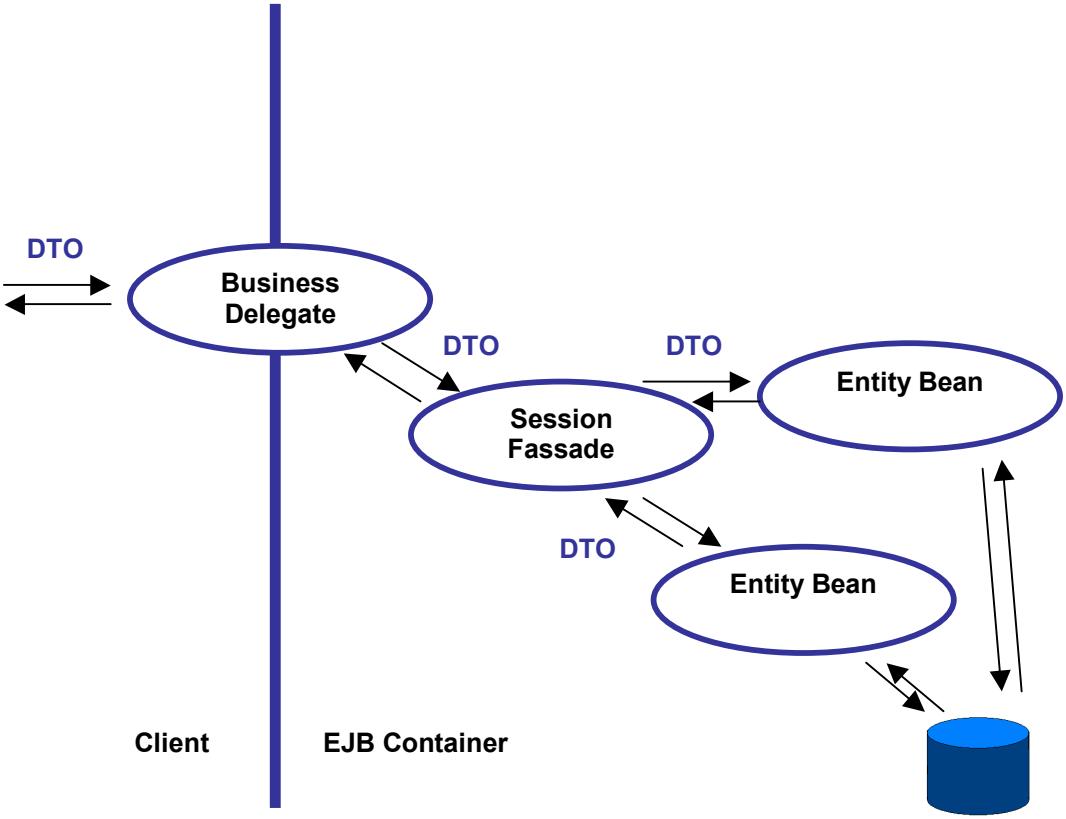


Abbildung 4: Komponenten der Design Patterns

Business Delegate

Alle Java Klassen, die die Endung BD haben, sind vom Pattern Business Delegate. Mit diesem Pattern soll der Zugriff der Clients auf die Geschäftslogik vereinfacht werden, die Zugriffslogik wird gekapselt, Cachingmechanismen können implementiert werden (was bei dem Projekt BAMS nicht eingesetzt wurde), die Wartbarkeit und Testbarkeit erhöhen sich und es minimiert die Kopplung zwischen den Clients und der Geschäftslogik.
Der Business Delegate wird als Schnittstelle zwischen der Präsentationschicht, also den `JSP` und Servlet-Seiten, und der `EJB`-Geschäftslogik definiert. Somit wäre die Implementierung der eingesetzten Delegate austauschbar. Im Business Delegate befindet sich keine Geschäftslogik. Dafür sind nur die SessionFassade und die Entity Beans im Projekt verantwortlich.

Session Fassade

Alle Java Klassen, die im Dateinamen Session Fassade beinhalten, sind von dem Pattern Session Fassade. Diese Fassaden sind Stateless Session Beans. Mit diesen Fassaden soll der Zugriff auf die Entity Bean gekapselt werden, es besteht die Möglichkeit des Cachens und intelligenten Transaktionssteuerungen, sowie eine Vereinfachung der Business Schnittstelle zum Business Delegate. Die Caching Mechanismen werden im Projekt nicht berücksichtigt. Um diese Mechanismen verwenden zu können muss die Fassade eine Stateful-Variante sein.

DTO

Alle Klassen, die DTO beinhalten, sind von dem Pattern DTO oder auch Value Object genannt. Das DTO Pattern stellt eine Datenkopie des entsprechenden Business Objekt dar, lässt fein granulare Zugriffe lokal auf der Client- Seite erfolgen und spart Remote Zugriffe auf die Business Schicht. Hierzu werden die Daten des Business Objekt über Setter dem DTO übergeben und der Client bekommt als Rückgabewert dieses DTO und kann über Getter die fein granularen Daten aus dem Objekt erhalten.

Transaktionsverhalten der Komponenten

Mit Transaktionen kann man verschiedene Aktionen bündeln und innerhalb einer Transaktion ausführen. Doch Transaktionen sind mehr als eine Aneinanderreihung von verschiedenen Aktionen, sie garantieren auch die Robustheit eines Verteilten Systems, wie das BAMS- Projekt. Die Robustheit eines Systems wird durch die ACID- Eigenschaften von Transaktionen erreicht. ACID steht für Atomicity (Unteilbarkeit), Consistency (Konsistenz), Isolation (Isolation von verschiedenen Aktionen) und Durability (Dauerhaftigkeit der Ergebnisse).
Wichtig bei dem Punkt Isolation sind die Isolationlevel der Transaktionen.
Die Isolationlevel bestimmen in welcher Form die Datenbank in Bezug auf konkurrierende Zugriffe gesperrt wird. Je höher das Level desto höher ist die Qualität der ausgelesen und gearbeiteten Daten aus der Datenbank, denn durch exklusiven Sperren der Datenbank können keinen anderen parallelen Transaktion die Daten ändern. Probleme wie Dirty Reads, Unrepeatable Read und Phantom Read werden dadurch ausgeschlossen, aber diese exklusive Sperrung bedeutet, dass die konkurrierenden Transaktionen auf die Freigabe warten müssen. Dies hat negative Einflüsse auf die Performance des Systems.
Für das Projekt wurde an den Standardeinstellungen der Datenbank für die Isolationlevel nichts geändert, das heißt, dass der Isolationlevel der Datenbank „READ COMMITTED“ ist. Dadurch werden die Dirty Reads eliminiert. Dies bedeutet aber trotzdem eine gute Performance aus der Datenbanksicht für das Projekt BAMS.

Die EJB 2.1 Spezifikation unterstützt nur flache Transaktionen, d.h. das eine Transaktion entweder committed oder rückgängig (Rollback) gemacht werden.
Für die Transaktionsabgrenzung für EJBs gibt es zwei Typen. Die deklarative und die programmatische Abgrenzung.
Um die Komponenten im Projekt mit einem Transaktionsverhalten zu versehen, wurde die deklarative Abgrenzung von Transaktionen gewählt, d.h. die Grenzen einer Transaktion werden im Deployment Deskriptor deklariert. Damit übernimmt der EJB- Container die Steuerung der Transaktion.

Im unteren Programmbeispiel ist eine deklarative Abgrenzung zu sehen. Diese werden in dem Deployment Deskriptor EJB-jar.xml eingestellt. Der Tag <container-transaction> beschreibt, dass der EJB-Container die Steuerung übernehmen soll. <EJB-name> gibt den Namen der EJB-Komponente an und <method-name>*</method-name>, dass alle Methoden der EJB das gleiche Transaktionsverhalten hat. Ganz wichtig ist der Tag <trans-

attribute>Required</trans-attribute>, der bestimmt welche Transaktionsattribute für die Methoden gelten sollen.

Insgesamt gibt es sechs Attribute:
NotSupported, Required, Support, RequiresNew, Mandatory, Never.
Diese Attribute bestimmen, wie die Transaktionskontexte übergeben werden. Das Transaktionsbeispiel Required bedeutet, dass eine Methode aus der Komponente Kunde immer in einem Transaktions-Kontext ausgeführt werden. Wenn der Client einen Transaktions- -Kontext mit übergibt, dann wird die Methode in diesem Kontext ausgeführt. Wenn aber der Client keinen Kontext mit übergibt, dann erzeugt der EJB- Container einen Kontext und beendet ihn nach dem Methodenaufruf wieder.

```
<assembly-descriptor>
    <container-transaction>
      <method>
        <EJB-name>Kunde</EJB-name>
        <method-name>*</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
      </container-transaction>
  </assembly-descriptor>
```

Transaktionsabwicklung mit den SessionFassaden
Durch die Sessionfassaden besteht die Möglichkeit, dass die Methoden, die im Business Delegate über die Sessionfassaden aufgerufen werden, in einem Transaktionskontext abgearbeitet werden können. In der Sessionfassade wird ein neuer Transaktionskontext neu erstellt und mit an die Entity Beans übergeben. Die Entity Beans übernehmen den Kontext mit dem deklarativen Befehl Required.
Somit ist die aufgerufene Methode in einer Transaktion gekapselt.

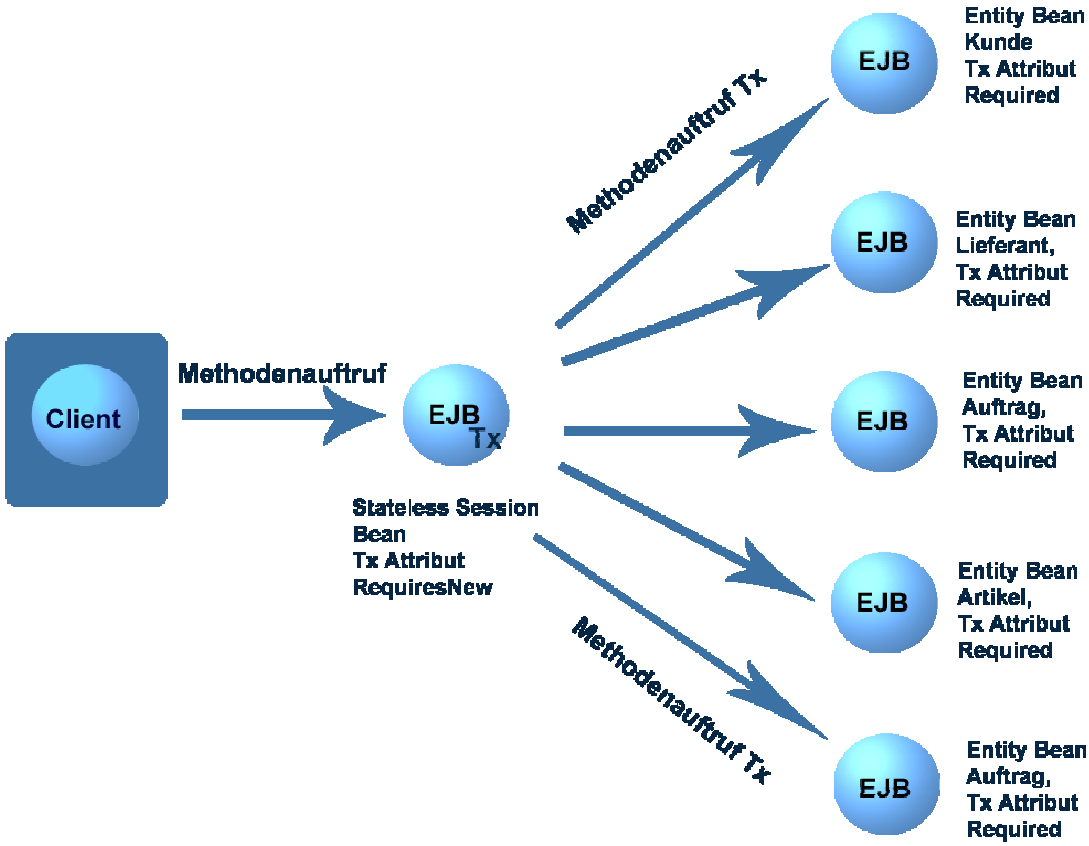


Abbildung 5: Transaktionsverhalten der Beans

EJB Security

Dieser Punkt ist in dem Projekt BAMS noch nicht verwirklicht. Es soll aber noch eine deklarative Sicherheit in den Beans verankert werden. Dazu werden Security Rollen und Method Permissions deklariert.

Applikationsserver

Eigenschaften

Jboss ist ein `J2EE` konformer Open Source Application Server.
Mit einer großen Anzahl von aktiven Entwicklern hat er einen hohen Verbreitungsgrad.
Veröffentlicht unter der `LGPL` Lizenz ist eine ernstzunehmende Alternative zu kommerziellen Produkten.
Es sind zahlreichen kostenlose Online Manuals und Tutorials verfügbar.
`JBoss` unterstützt verschiedene Java-Standards wie `EJB 3.0` und Java-Technologien, wie z.B.

```
JDBC
Servlets/ JSPs
JCA
JMS
JNDI
...
```

Deployment

Des Weiteren verfügt der JBoss Server über die Möglichkeit des „Hot- Deployment“:
- der laufende Server muss für das Deployment nicht gestoppt werden
- Undeploy und Redeploy sind möglich

Es können mehrere Datentypen deployed werden:
- `EJB JARs`
- `EAR` (Enerprise Archives)
- `WAR` (Web Archives)
- `RAR`
- `SAR` (Service speziell für JBoss)

Architektur

Der JBoss hat einen modularen Aufbau mit Mikrokernel „ohne“ eine eigene Funktionalität, diese wird über Modul Plugins „eingefügt“. Dies wurde mittels den Java Management Extensions (JMX) von Sun realisiert.

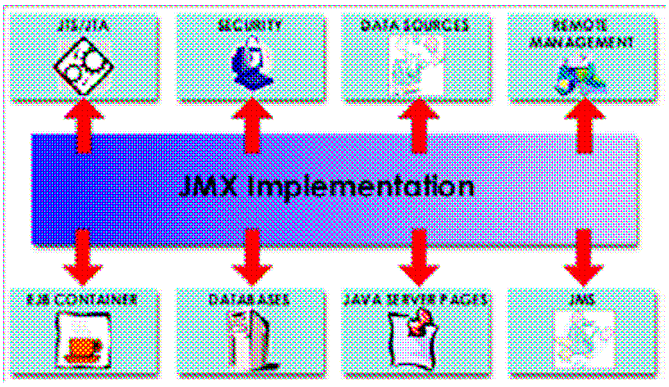


Abbildung 8: JMX mit modularen Plugins

Installation

• Zunächst muss der Applikationsserver von `http://www.jboss.org/downloads/index` heruntergeladen werden.
• Damit der Server verwendet werden kann wird mindestens das `JDK 1.4` benötigt. Falls man lieber nur das `JRE` verwendet, kann es während der Kompilierung der JSPs zu Problemen kommen.
• In die `JAVA_HOME` Umgebungsvariable wird der Ort der `JDK` Installation gesetzt

Serverstruktur

Um den JBoss für eigene Applikationen und Anwendungen verwenden zu können muss man sich unbedingt mit der Verzeichnisstruktur des Applikationsservers auskennen.

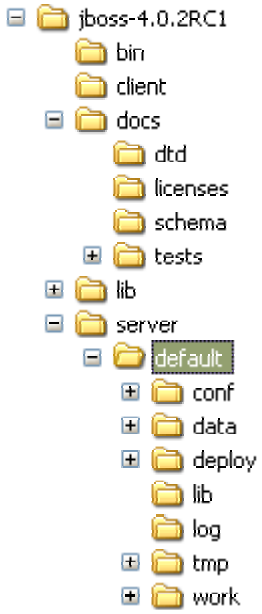


Abbildung 9: JBoss Verzeichnissbaum

• bin : beinhaltet System spezifische Scripte u.a. auch um den Server zu starten und wieder stoppen.
• client : verwaltet Konfigurations- und JAR Dateien, welche von externen Webcontainern oder Java Clients benötigt werden
• docs : beinhaltet die XML DTDs, welche im JBoss für die Referenzierung verwendet werden. Ebenso gibt es verschiedene Beispiele in Form von JCA Konfigurations Dateien (Java Connector Architekture) um Daten in verschiedene Datenbanken (z.B MySQL, Oracle,) aufzusetzen.
• lib : enthält JAR Dateien, welche für den JBoss Mikrokernel benötigt werden. Hier sollten keine eigenen JAR Dateien eingefügt werden!!
• server : jedes Unterverzeichniss hier ist eine andere Serverkonfiguration.
Diese werden noch genauer beschrieben.

Starten des JBoss

JBoss wird über die Datei run.bat im „`bin`"- Verzeichnis der default- Serverkonfiguration gestartet.

Stoppen des JBoss

JBoss kann über die Datei shutdown.bat im „`bin`"- Verzeichnis der default- Serverkonfiguration gestoppt werden oder über die Tastenkombination STRG – C.

Serverkonfiguration

Im JBoss sind bereits drei Serverkonfigurationen vorhanden – `all, default, minimal.`
Diese Einstellungen bieten jeweils verschieden Services an.
In diesem Projekt wird die default Serverkonfiguration verwendet, wie auch von den meisten J2EE-Applikationen.
Diese Konfiguration beinhaltet keinen JAXR -, IIOP oder irgendeinen Clustering Service.
Die unterstützten Services der beiden anderen Serverkonfigurationen können im "Getting Started with Jboss 4.0" auf `www.jboss.org` nachgelesen werden.
Es können auch eigene Serverkonfigurationen definiert werden, welche im server-Verzeichniss abgelegt werden.

Einige wichtige Bestandteile des server Verzeichnisses kurz erklärt:

deploy : Einer der wichtigsten Teile des Server Verzeichnisses ist der deploy Ordner. Hier werden die Applikationsdateien(JAR, WAR und EAR) reingelegt, welche deployed werden sollen. Dieses Verzeichniss wird regelmässig auf Änderungen überprüft, so dass veränderte Komponenten automatisch re-deployed werden. Ebenfalls wird es für hot- deployed Services verwendet. Es können während der Server läuft neue Komponenten hinzugefügt werden.

lib: beinhaltet die von dieser Serverkonfiguration benötigten JAR- Dateien. Benötigte Bibliotheken, wie z.B. der JDBC Treiber werden hier eingefügt.

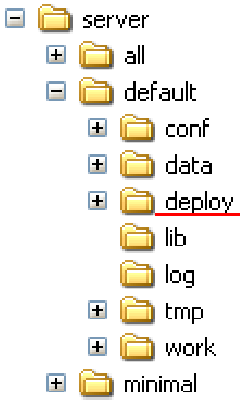


Abbildung 10: default – Serverkonfiguration

Die Verzeichnisse `data, log, tmp, work` werden von JBoss erzeugt. Dazu muss der Server mindestens einmal gestartet worden sein.

Stolpersteine

Die Einarbeitungszeit in die Architektur und die Funktionsweise des JBoss Applikationsservers hat einen sehr großen Teil des Projektes in Anspruch genommen.
Besonders das Finden, Umsetzten und Verstehen aller Einstellungen war sehr aufwendig ebenso wie die Auseinandersetzung und das Verstehen der Fehlermeldungen.

4.3. Backend-Tier

Datenbanktabellen

Um das Projekt zu realisieren wurde folgende Datenbankarchitektur gewählt.

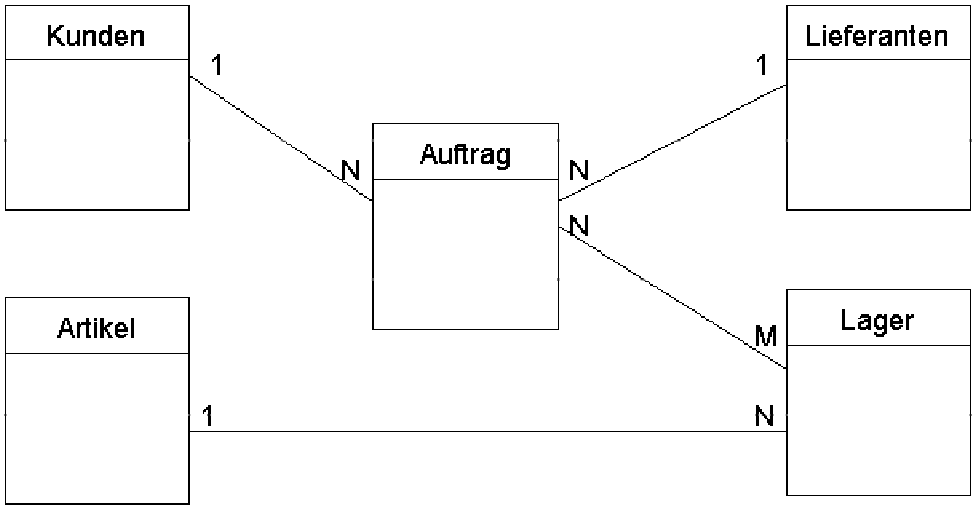


Abbildung 6:Datenbanktabellen

Es wurden insgesamt 6 Datenbanktabellen entwickelt. Diese stehen in Beziehungen zu einander. Es gibt Eins-zu-Viele Beziehungen und zwischen dem Auftrag und dem Lager gibt es eine Viele-zu-Viele Beziehung. Diese Beziehung muss in zwei Eins-zu-Viele Beziehungen aufgesplittet werden.

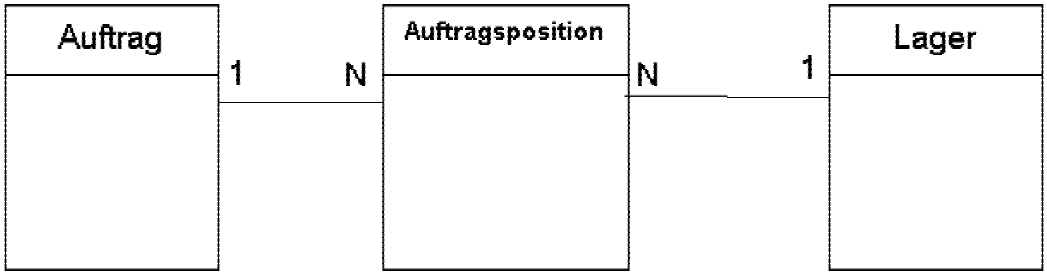


Abbildung 7:Aufgesplitetes n:m Verhältniss

Datenbankkonfiguration

Die einzelnen Schritte der Datenbankkonfigurtion für das Projekt BAMS und den Applikationsserver jboss werden hier aufgeführt.

In der MySQL- Datenbank:

1.Neue Datenbank erstellen Name jboss

```
CREATE DATABASE jboss;
```

2.User mit Passwort erstellen : user bams, passwort weiberg

```
GRANT ALL PRIVILEGES ON jboss.* TO bams@localhost IDENTIFIED BY
'weiberg'
```

Im Applikationsserver jboss

3.JDBC Treiber installieren

Der JBoss muss die JDBC Treiber Klassen kennen um mit der Datenbank kommunizieren zu können. Für die mySQL Datenbank wird die Datei `mysql-connector-java-3.0.15-ga-bin.jar` benötigt. Diese Datei findet man unter `http://dev.mysql.com/downloads/connector/j/3.1.html`.

4.XML-Datei

Im Ordner `examples/docs/jca` findet man die entsprechende XML-Datei für die mySQL Datenbank, in welcher man lediglich die Datenbank, den Usernamen und das Passwort ändern muss. Die Datei mysql-ds.xml muss danach in den Ordner deploy des Servers kopiert werden. Falls es bereits eine Datei mit der Endung `-ds.xml` gibt muss diese gelöscht werden. Vorsicht: nicht in einen anderen Ordner des Servers kopieren sondern wirklich löschen!

Stolpersteine

Falls der `JBoss` Applikationsserver noch keine Verbindung zur Datenbank aufbauen sollte, sollten wirklich alle Standard `-ds.xml` Dateien aus den gesamten `JBoss` Verzeichnissen in die `mysql-ds.xml` Datei geändert werden.

5.zusätzliche Features

5.1.Übernahme der bestehenden Daten

Problem

Die Kunden- und Lieferantendaten, sowie die Informationen der Artikel wurden bereits in einer „Faktura"- Datenbank in je einer Tabelle gehalten. Dies waren insgesamt über 900 Datensätze.
Um diese Daten automatisch in die vom JBoss verwendete MySQL Datenbank zu schreiben musste eine Lösung gefunden werden.

Lösung

Die Daten der drei Tabellen wurden zunächst in eine Excel-Datenbank exportiert.
Von dort aus konnten sie jeweils als Komma Seperated File (csf) dargestellt werden.
Diese Text-Dateien wurden zeilenweise ausgelesen und die Datensätze in die richtigen Tabellen der MySQL- Datenbank geschrieben.

Verwendete Dateien

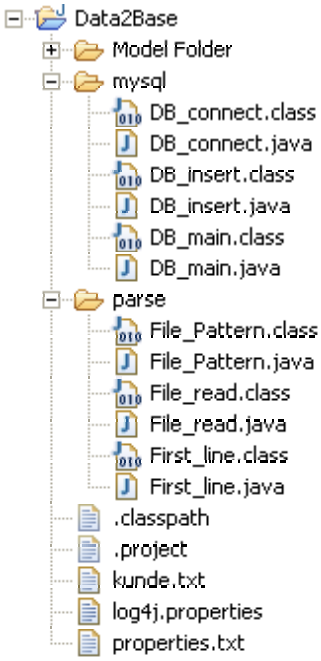


Abbildung 11: Dateien zu Übernahme der Altdaten

Package mysql
Beinhaltet Klassen um die Verbindung zur JBoss-Datenbank zu realisieren und die Statements durch den Aufruf der `file2DB`-Methode einer `File_read`-Instanz in die definierten Tabellen zu schreiben.

Package parse
Die Klassen hier haben zwei Funktionen.
Zum einen lesen sie die erste Zeile der angegebenen Textdateien aus und geben die Namen der Tabellenspalten und die Anzahl der Spalten aus. Um die einzelnen Tabellenspalten zu erhalten werden reguläre Ausdrücke zu Hilfe genommen.
Außerdem werden die Datensätze zeilenweise in die Tabellen geschrieben.

Weitere Dateien
Im `properties.txt` steht das Pattern der regulären Ausdrücke.
`Log4j.properties` beinhaltet die Definierten Eigenschaften für das logging.
`Kunde.txt, Lieferanten.txt`, Artikel.txt enthalten die Daten der jeweiligen Tabellen als Komma seperated Files. Diese Dateien stehen für die Projektübergabe jedoch nicht zur Verfügung, da es sich um betriebsinterne Daten der Firma „Weiberg“ handelt.

Stolpersteine

Zunächst waren diese Klassen dafür vorgesehen, die gesamten Textdateien mit Hilfe von regulären Ausdrücken durchzugehen, jeden Datensatz einzeln auszulesen und in das richtige Feld in der Datenbank einzusetzen. Dies scheiterte daran, dass die Daten einzelner Spalten keine einheitliche Form hatten und auch in den Datensätzen selbst sämtliche Sonderzeichen vorkamen. So dass nun die gesamte ausgelesene Zeile als String in ein Statement umgewandelt wird. Dies bedeutet, dass die Tabellen in der MySQL- Datenbank wieder genauso aufgebaut sind wie die in der alten Datenbank.

5.2. Personalisierte Rechnung

Anforderungen

Nachdem man die bestellten Artikel eines Kunden ausgewählt hat sollte man sich auch eine Rechnung über Bestellung für den jeweiligen Kunden anzuzeigen oder abzuspeichern. Die Rechnung sollte als Email-Anhang verschickt werden können. Ebenso, was eine der wichtigsten Anforderungen war, sollte sie ausgedruckt werden können.
Auch die Möglichkeit von mehrseitigen Rechnungen muss gegeben sein.

Lösung

Für die Umsetzung der Anforderungen konnte man zwei Wege in Betracht ziehen.
Es gab die Möglichkeit einer für den Ausdruck modifizierten html-Datei (ein für den Druck optimierte xsl-Datei).
Ein weitere Option war die Darstellung in einem pdf- Format.
Die Entscheidung fiel auf die Ausgabe der Rechnung als pdf, da dieses Format allen oben genannten Anforderungen gerecht werden konnte.

Verwendete Dateien

XML Generierung
Ein `XML`- Dokument wird für jede Rechnung mit den übergebenen Werten neu generiert.
Zur Realisierung der `XML`- Struktur wird die Darstellung als `jdom` verwendet.
Die bestehende `XML`- Datei wird jedes Mal von einer neuen überschrieben.

PDF Generierung
Die Klasse `pdfGenerierung` wird verwendet um aus dem `XML`- Dokument und `XSLT`-Dokument ein Dokument im pdf- Format zu erzeugen. Hierzu wird der java Opensource `XSLT`-Prozessor `FOP (Formatting Object Prozessor)` verwendet.

weitere Dateien
`rechnungFO.xsl` beinhaltet die **Eigenschaften für die Darstellung** der Rechnung. Diese `XSL`-Datei wird nicht verändert.

Durchführung
Vorgehensweise innerhalb des ApacheFOP bei einer Transformation von xml zu `pdf`:
Zunächst wird aus der `xml`- und `xsl`- Datei ein formatting object Dokument generiert, welches dann durch den `xslt`- Prozessor zu einem `pdf`- Format umgewandelt wird.

Abbildung 13:Vorgehensweise bei der Transformation von xml zu pdf

Ergebnis

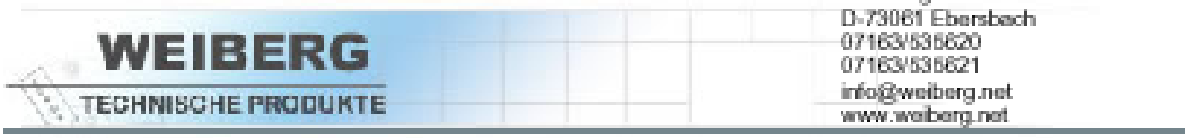
- Als Datum wird immer der Tag der Erstellung der Rechnung verwendet.
- Die zu Summe der einzelnen Produktpreise wird ausgerechnet.
- Die Mehrwertsteuer wird hinzugefügt und der zu zahlende Bruttobetrag errechnet.
- Der Header und Footer bleiben immer gleich, falls eine Seite nicht mehr ausreicht.
- Die Nummerierung der Seiten geschieht automatisch.

Damit es nicht zu Verwechslungen der Rechnungen für Kunden kommt wird der Dateiname des pdf- Formats automatisch aus der Rechnungsnummer und dem Kundennamen erzeugt. So, dass die Rechnungen eines Kunden nicht durch aktuellere überschrieben werden können.

Stolpersteine

Der ApacheFOP ist nicht in der Lage alle möglichen `xsl:fo` Ausdrücke richtig umzusetzen.

Vor allen Dingen bei Tabellen, die laut Richtlinien wie folgt umgesetzt werden, kann der ApacheFOP nicht mit `<fo:table-and-caption>`. Hier wird die Tabelle dann gar nicht angezeigt. Also wird `<fo:table-and-caption>` einfach weggelassen und der Tabellenaufbau mit `<fo:table>` begonnen.



Weiberg technische Produkte
Kornbergstraße 8
D-73061 Ebersbach
07163/536820
07163/536821
info@weiberg.net
www.weiberg.net

Heidrun
Schmied
Feuergasse 6
34533 Fernhumm
MaiLand

6.Mai 2005

Kundennummer 23463-3457

Rechnungsnummer 1344534

Herzlichen Dank für ihr Vertrauen in unsere Dienstleistungen. Wir erlauben uns folgende Beträge in Rechnung zu stellen und freuen uns wenn Sie sich wieder bei uns melden.

POS	Bezeichnung	EP	Menge	Gesamt
3451453	Bandstreichel Schleifer 827-DA34	553.34	6	3320.04
877899	Spiral-Bohrer 234DV-23	433.66	3	1300.68
1346134	Drehschleifer ohne Achse 34K-32	23.44	2	46.88
3451453	Dynamischer Hammer 234JK2-23	253.69	10	2536.90
				7204.50
		Netto		7204.50
		16,00% MwSt		1152.72
		Gesamtbetrag		**8357.22**

Wir möchten Sie bitten, den offenen Betrag an unten stehende Bankverbindung zu überweisen. Bei der Zahlung mittels Internet-Banking tragen Sie bitte im Feld "Verwendungszweck" die Rechnungsnummer ein.

Deutsche Bank München
Bankleitzahl: 23004002 | Kontonummer: 21341233
BIC/SWIFT: DEUTDEDBMUC
IBAN: DE20700700246400456867000070

Seite 1

Abbildung 14:personalisierte Rechnung

6. Offene Punkte

- Integration der Lagerkomponente
- ein kundenorientiertes angepasstes Exception Handling, damit sich die Firma „Weiberg“ nicht mit den Fehlermeldungen von System auseinander setzten muss. Dies wird umgesetzt indem die Systemfehlermeldungen im Business Delegate Pattern abgefangen werden und in benutzerfreundliche Fehlermeldungen umgesetzt werden. Dies geschieht durch Öffnen eines seperaten Fensters(popup), in dem eine für den Benutzer verständliche Fehlermeldung erscheint wie z.B „Rufen Sie Ihren Admin an“/“Fassen Sie bloß nichts mehr an“ :-).

- Security – also Autorisierung und Autetifizierung, Übergabe der Credentials, Haltung der Credentials. Diese Punkte werden noch u.A. mit der Verwendung von JAAS umgesetzt.

7. Erleichterungen für Programmierer

verwendete Plugins

- QuantumDB

  Plugin zur Verwaltung von Datenbanken :
  - arbeitet mit jeder JDBC-konformen Datenbank
  - Schemas, Views, Sequenzen anschauen
  - Tabellen anzeigen lassen
  - Manipulation der Datenbank
  - und noch weitere Features
  - Download unter http://sourceforge.net/project/

- MyEclipse

  Plugin für J2EE-Deployment:
  - unterstützt JSPs, Servlets, Struts, EJBs, JSF und Hibernate
  - intergriert den jeweils eingesetzten Applikationsserver
  - full-lifecycle Support für Programmierung, Deploy, Test, und Debug

Benötigte Bibliotheken

-jdom
-fop
-log4j-1.2.9
-avalon-framework-cvs-20020806
-xalan-2.4.1
-batik

8. Quellen

• JBoss Homepage
http://www.jboss.org

•Getting Started with Jboss 4.0 / Release 3

• JBoss auf Sourceforge
http://sourceforge.net/projects/jboss

• Java 2 Enterprise Edition Plattform
http://java.sun.com/j2ee

• http://www.xslfo.info

•Enterprise JavaBeans komplett
Grundlagen, Überblick und Einsatz von EJB 2.1
O.Ihns(Hrsg.)/S.Heldt/R.Wirdemann/H.Zuzmann