

**Endbericht  
PG 551: "Excelerate"**

**Ettiboa Adouakou, Georgios Christidis, Alexander Drees, Felix Gabriel, Christian Hammerl, Kersten Lorenz, Donato Marro, Jan Pardo, Carola Thalmann, Julia Werdelmann, Hongzhi Wang**

10. Oktober 2011



**Institution**

Technische Universität Dortmund  
Fakultät für Informatik  
Lehrstuhl 5 für Programmiersysteme

**Betreuer**

Dipl.-Inform. Markus Doedt  
Dipl.-Inform. Sven Jörges  
Dr. Ralf Nagel



# Inhaltsverzeichnis

<b>Abkürzungsverzeichnis</b>	<b>v</b>
<b>1. Vorwort</b>	<b>1</b>
<b>2. Einführung</b>	<b>3</b>
2.1. Problemstellung und Motivation . . . . .	4
2.2. Inhalte und Ziele der Projektgruppe . . . . .	5
2.3. Aufbau des Endberichts . . . . .	6
<b>3. Grundlagen</b>	<b>9</b>
3.1. Java EE . . . . .	9
3.2. EJB . . . . .	10
3.2.1. Motivation . . . . .	10
3.2.2. Session Beans und Message-Driven Beans . . . . .	11
3.2.3. Entity Beans . . . . .	12
3.2.4. Transaktionen . . . . .	13
3.2.5. JPA . . . . .	14
3.3. Webservices . . . . .	15
3.3.1. Webservice-Technologie . . . . .	15
3.3.2. JAX-WS Projekt . . . . .	17
3.4. Tapestry . . . . .	17
3.4.1. Konvention über Konfiguration . . . . .	17
3.4.2. MVC-Entwurfsmuster (MVC-Pattern) . . . . .	18
3.4.3. Seiten (Pages) . . . . .	19
3.4.4. Tapestry Markup Language . . . . .	20
3.4.5. Komponenten . . . . .	20
3.4.6. Event-Behandlung . . . . .	21
3.4.7. Nachrichten-Kataloge . . . . .	22
3.4.8. Validierung . . . . .	23
3.4.9. Persistierung . . . . .	23
3.5. jBPM . . . . .	24
3.5.1. Process Virtual Machine . . . . .	25
3.5.2. Context . . . . .	25
3.5.3. Persistenz . . . . .	26
3.5.4. jPDL . . . . .	26
3.5.5. Graphical Process Designer . . . . .	28

3.5.6.	Beispielprozess . . . . .	28
3.6.	Scrum . . . . .	31
3.6.1.	Rollen in Scrum . . . . .	32
3.6.2.	Artefakte in Scrum . . . . .	33
3.6.3.	Ablauf . . . . .	34
<b>4.</b>	<b>Projektmanagement</b>	<b>37</b>
4.1.	Termine . . . . .	37
4.2.	Vorgehen à la Scrum . . . . .	38
4.3.	Konkretes Vorgehen . . . . .	39
4.4.	Redmine . . . . .	39
4.5.	Konventionen . . . . .	40
4.5.1.	Code-Konventionen . . . . .	40
4.5.2.	VCS-Konventionen . . . . .	41
<b>5.</b>	<b>Architektur</b>	<b>43</b>
5.1.	Schichtenmodell . . . . .	43
5.2.	Einrichtung . . . . .	43
5.3.	Testen . . . . .	45
<b>6.</b>	<b>Projekte</b>	<b>47</b>
6.1.	Projekt 1: Lagerverwaltung . . . . .	47
6.1.1.	Beschreibung . . . . .	47
6.1.2.	Ablauf . . . . .	48
6.1.3.	Zusammenfassung . . . . .	53
6.2.	Projekt 2: Stundenzettel . . . . .	54
6.2.1.	Beschreibung . . . . .	54
6.2.2.	Ablauf . . . . .	58
6.2.3.	Zusammenfassung . . . . .	69
6.3.	Projekt 3: Getränkeverwaltung . . . . .	69
6.3.1.	Beschreibung . . . . .	69
6.3.2.	Ablauf . . . . .	74
6.3.3.	Zusammenfassung . . . . .	100
<b>7.</b>	<b>Kritik</b>	<b>103</b>
<b>8.</b>	<b>Zusammenfassung</b>	<b>105</b>
<b>Anhang</b>		<b>I</b>
A.	Administratorhandbuch . . . . .	I
<b>Literaturverzeichnis</b>		<b>XXI</b>

# Abkürzungsverzeichnis

**API** Application Programming Interface  
**BLOB** Binary Large Object  
**BPEL** Business Process Execution Language  
**BPMS** Business Process Management System  
**EJB** Enterprise JavaBeans  
**GPD** Graphical Process Designer  
**GUI** Graphical User Interface  
**HSQldb** HyperSQL Datenbank  
**HTML** Hypertext Markup Language  
**HTTP** Hypertext Transport Protokoll  
**JAAS** Java Authentication and Authorization Service  
**Java EE** Java Platform, Enterprise Edition  
**Java SE** Java Platform, Standard Edition  
**JAX-WS** Java API for XML - Web Services  
**JBPM** Java Business Process Management  
**JDBC** Java Database Connectivities  
**JPDL** Java Process Definition Language  
**JPA** Java Persistence API  
**JSP** JavaServer Pages  
**JTA** Java Transaction API  
**LGPL** GNU Lesser General Public License  
**MVC** Model-View-Control  
**ORM** Objekt/relationales Mapping  
**PG** Projektgruppe  
**POI** Poor Obfuscation Implementation  
**POJO** Plain Old Java Object  
**PVM** Process Virtual Machine  
**RDBMS** Relational Database Management System  
**REST** Representational State Transfer  
**RMI** Remote Method Invocation  
**SOA** Serviceorientierte Architektur  
**SOAP** Simple Object Access Protocol  
**SQL** Structured Query Language  
**SVN** Subversion  
**UDDI** Universal Description, Discovery and Integration  
**VCS** Version Control System  
**W3C** World Wide Web Consortium

**WSDL** Web Services Description Language  
**XHTML** Extensible Hypertext Markup Language  
**XML** Extensible Markup Language  
**XSL** Extensible Stylesheet Language

# 1. Vorwort

Die Durchführung von Projektgruppen ist Bestandteil der Studiengänge Informatik und Angewandte Informatik der Technischen Universität Dortmund. Eine Projektgruppe (PG) besteht dabei, laut Vorgabe, aus acht bis zwölf Teilnehmern. Unter der Führung von zwei Betreuern werden über zwei Semester hinweg vielschichtige Aufgaben im Team bearbeitet. Das gibt sowohl Gelegenheit zur Übung in Teamarbeit und Koordination, als auch Anwendung bisher erlernter Methoden der Informatik. So kann über eine längere Dauer hinweg an komplexen Problemstellungen praktisch gearbeitet und ein wenig auf die postakademische Zeit vorbereitet werden.

Im Zeitraum des Wintersemesters 2010/2011 bis zum Ende des Sommersemesters 2011 wurde vom Lehrstuhl 5 der Fakultät Informatik dementsprechend die Projektgruppe „Excelerate“ angeboten. Thema des Projekts war die Entwicklung und Migration betriebswirtschaftlicher Software von Microsoft Excel-basierten Lösungen hin zu agilen, serviceorientierten Architekturen.

Der vorliegende Projektendbericht beschreibt die Zielsetzung, den Verlauf und das Ergebnis der Projektgruppe. Unser besonderer Dank gilt Prof. Dr. Bernhard Steffen, unter dessen Schirmherrschaft das Projekt stattgefunden hat, sowie unseren drei Betreuern Dipl.-Inform. Markus Doedt, Dr. Ralf Nagel und Dipl.-Inform. Sven Jörges (im Austausch für Dr. Ralf Nagel nach Ende des ersten Semesters), die uns immer mit Rat und Tat zur Seite standen.





## 2. Einführung

Eine Folge von Einzeltätigkeiten, die schrittweise ausgeführt werden um ein geschäftliches oder betriebliches Ziel zu erreichen, nennt man im Fachjargon Geschäftsprozesse. Diese müssen abteilungs- und unternehmensweit organisiert werden, damit ein reibungsloser und kosteneffizienter Ablauf überhaupt möglich wird. Geschäftsprozesse können dabei Teil anderer Geschäftsprozesse sein, oder weitere Geschäftsprozesse beinhalten oder diese anstoßen. In diesem Zusammenhang setzen alle Unternehmen heute verschiedenste Softwareprodukte ein.

Zur Standardausrüstung eines Unternehmens gehören zweifelsohne auch die verschiedenen Applikationen einer Office-Lösung, wie sie Microsoft Office [Microsoft Corporation, 2011b] oder OpenOffice.org [OpenOffice.org] bereitstellen. Diese Softwarepakete unterstützen die täglichen Arbeiten eines jeden Büromitarbeiters und sind für die meisten Angestellten unersetzlich.

Neben den komfortablen Textverarbeitungsprogrammen, die zum Schreiben von Dokumenten jeglicher Art benötigt werden, gehören auch Tabellenkalkulationen zum Umfang eines solchen Softwareprodukts. Hier können Daten nicht nur aufbereitet und angezeigt werden, auch verschiedene Sortier-, Gruppier- und Filterfunktionen stehen zur Verfügung.

Die meisten Tabellenkalkulationen ermöglichen außerdem umfangreiche Berechnungen auf Basis individueller Formeln. Bereits integrierte Funktionen helfen, kaufmännische oder statistische Kalküle zu verwenden. Auch der Umgang mit komplexen Datumsfunktionen ist gegeben. Ebenso können Daten verarbeitet - zum Beispiel Texte verkettet - werden. Logische Berechnungen (if-else) zur Entscheidungsverzweigung gehören auch zu ihrem Repertoire. Ein weiteres Merkmal sind Visualisierungsmöglichkeiten in Form von Diagrammen und Bildern.

Microsoft Excel [Microsoft Corporation, 2011a], als Bestandteil von Microsoft Office, ist hier der Quasi-Standard und wahrscheinlich auf sehr vielen Bürocomputern installiert. So gut wie jeder Computernutzer ist wohl schon einmal in Berührung mit diesem Programm gekommen. Ähnliche Programme, wie OpenOffice Calc, orientieren sich stark an Microsoft Excel und stellen selbst die Bearbeitung von Exceldateien zur Verfügung.

Somit ist auch klar, warum sich Tabellenkalkulationen im Umfeld von Geschäftsprozessen mehr und mehr durchsetzen [Software-Initiative Deutschland e.V., 2011]. Sie bieten umfangreiche und flexible Möglichkeiten diese Prozesse zu organisieren und zu unterstützen. Außerdem sind sie leicht zu bedienen und hochverfügbar. Jeder Mitarbeiter, der wiederkehrende Berechnungen und Geschäftsprozesse zu bearbeiten hat, bedient sich erfahrungsgemäß einer solchen Tabellenkalkulation.

## 2.1. Problemstellung und Motivation

Der extensive Gebrauch, aufgrund der erwähnten Vorteile einer Tabellenkalkulation, führt aber auch zu Problemen. So wird die Software meist überstrapaziert und zweckfremdet. Aufgaben, für die sie eigentlich gar nicht konzipiert ist, werden mit ihr übernommen. Außerdem entziehen sie sich der Kontrolle durch andere Mitarbeiter. Wer weiß schon welche Exceldateien im eigenen Unternehmen in Gebrauch sind?

Gerade neue Mitarbeiter oder Benutzer, die eine solche Tabelle zum ersten Mal sehen und damit arbeiten müssen, haben Probleme die Inhalte zu verstehen. Die Reihenfolge der Bearbeitung kann beispielsweise eine Rolle spielen. Implizites Wissen ist also notwendig. Und solch fehlendes Wissen provoziert Fehler. Ein solches Dokument wächst zudem mit der Zeit und wird eventuell sogar von mehreren Personen gepflegt und erweitert. Dadurch steigt natürlich die Komplexität und eine falsche Bedienung wird weiter forciert. Das führt zu falschen Berechnungen und Inkonsistenzen, welche die Fehleranfälligkeit steigern. Eine vernünftige Dokumentation würde hier Abhilfe verschaffen, diese liegt aber meist gar nicht vor und die Anwendung bleibt schwer nachvollziehbar und unübersichtlich. Möglicherweise verweilt der Schöpfer der Datei auch gar nicht mehr im Unternehmen. So bleiben Sinn, Zweck und Vorgehen noch mehr im Dunkeln.

Mit Sicherheit deckt eine Tabellenkalkulation wie Excel auch nicht alle Funktionen ab, die zu einer eleganten Geschäftsprozessmodellierung notwendig wären. Zum Beispiel fehlen Faktoren zur Steuerung automatischer Abläufe oder dem Einbinden externer Services zu anderen eingesetzten Programmen im Unternehmen. Ungenutztes Potenzial entsteht aber vor allem auch - neben dem Fehlen ganzer Funktionalitäten - durch Unkenntnisse. Schließlich werden diese Tabellen meist von Personen entwickelt, die zwar über alle kaufmännischen und betriebswirtschaftlichen Hintergründe verfügen, sich aber auf dem Feld der Informationsverarbeitung nur bedingt auskennen. Denn unter der Haube ist Microsoft Excel zweifellos ein mächtiges Werkzeug, das sogar über eine integrierte Programmiersprache verfügt (VBA). Aber ohne ein Studium der Informatik oder vergleichbarer Ausbildungen bleiben diese Merkmale oft unentdeckt oder unbenutzbar. Um beispielsweise ein Mehrbenutzersystem mit Hilfe von Microsoft Excel aufzubauen bzw. zu ermöglichen, sind fundierte Kenntnisse des Programms vonnöten. So behilft man sich hier meist ein paar „schmutziger Tricks“, die ihr Übriges zur Fehleranfälligkeit beitragen.

Auch das Nachhalten einer Chronik erweist sich in Excel als schwierig. In den Tabellen werden oft alte Werte durch neue überschrieben und man weiß nachher nicht mehr, was vorher berechnet wurde. Zur Lösung kann man die Exceldateien natürlich einfach beliebig oft kopieren und bei jeder Änderung eine neue Kopie anlegen. Besser wäre es aber die alten Daten sauber in dafür vorgesehenen Datenbanktabellen zu hinterlegen und entsprechende Dienste zu definieren, die diese Daten dem Benutzer zugänglich machen.

Der Charakter einer solchen Anwendung bleibt im Laufe ihres mitunter langen Lebens deshalb oft prototypisch. Im Laufe der Zeit entstehen große Lösungen, die zwar

irgendwie funktionieren aber umständlich zu bedienen sind, fehleranfällig sind oder das Potenzial nicht voll ausschöpfen. Dennoch können diese Dokumente einen entscheidenden Beitrag leisten, wenn es darum geht den so modellierten Geschäftsprozess in eine vernünftige Reinform zu bringen. Denn sie bilden eine gute Basis und einen Startpunkt für weitere Untersuchungen.

## 2.2. Inhalte und Ziele der Projektgruppe

Es ist also sinnvoll solche problematischen Strukturen in vernünftige Software zu übersetzen. Das gilt vor allem für Exceltabellen, die Geschäftsprozesse simulieren und erheblich vereinfacht werden könnten. Doch wie kann eine solche „Problemstellung“ in eine adäquate Software umgesetzt werden? Und welcher Unterbau ist für eine solche Umsetzung geeignet, sprich welche Architektur dient als Grundlage für solche Migrationen? Genau diese Fragestellungen bilden den Kernpunkt der Arbeit dieser Projektgruppe.

Im Bereich der Geschäftsprozessmodellierung haben sich in letzter Zeit moderne, serviceorientierte Architekturen [Aier u. Schönherr, 2006] durchgesetzt. Eine Serviceorientierte Architektur (SOA), auch dienstorientierte Architektur, ist ein Architekturmuster der Informationstechnik aus dem Bereich der verteilten Systeme, um Dienste von IT-Systemen zu strukturieren und zu nutzen. Durch Zusammensetzen (Orchestrierung) von Services niedriger Abstraktionsebene können recht flexibel, unter Ermöglichung größtmöglicher Wiederverwendbarkeit, Services höherer Abstraktionsebenen geschaffen werden. Die Komplexität der einzelnen Anwendungen bleibt dabei hinter den standardisierten Schnittstellen verborgen.

Diese Dienste werden meist durch verschiedene Protokolle über das Internet zur Verfügung gestellt. Hierbei sollte beachtet werden, dass nicht jeder Dienst immer selbst implementiert werden muss. Es gibt Lösungen, die bereits fertig implementiert sind und sich einfach als Service in eine SOA integrieren lassen.

Diese Architekturen bestehen oft aus vier Schichten: Der Datenbankschicht, der Serviceschicht, der Prozessschicht und der Präsentationsschicht (siehe Kapitel 5). Die Datenbankschicht bildet die Grundlage der Datenhaltung und besteht aus den Tabellen bzw. der Datenbank. Darauf bauen die Services auf, die diese Daten nach außen anbieten und sie entsprechend manipulieren können. Zusätzlich bündelt die Prozessschicht diese Dienste und stellt durch entsprechende Tools Möglichkeiten bereit, die komplexen Geschäftsprozesse zu orchestrieren. Die Präsentationsschicht stellt dem Benutzer des Systems eine grafische Oberfläche zur Verfügung anhand der er durch die diversen Geschäftsprozesse geführt wird.

Ziel der Projektgruppe war es, einen solchen Aufbau (in Anlehnung an diese Architektur) zu konstruieren und anhand mehrerer Problemstellungen zu erkunden. Dazu gehörte die Einarbeitung in die entsprechenden Themen, sowie die Durchleuchtung des gesamten Arbeitsablaufs von der Analyse einer bestehenden Exceltabelle und den damit verbundenen Prozessen über weiterführende Interviews bis hin zur fertigen serviceorientierten Architektur in Form einer Webanwendung. Ge-

rade die Prozessschicht gilt an dieser Stelle nochmals gesondert hervorzuheben, da sie von höchster Bedeutung für die Inhalte unserer PG war. Die Modellierung von Geschäftsprozessen mittels adäquater Mittel - wir setzten *Java Business Process Management (jBPM)* (siehe Abschnitt 3.5) ein - war ein Schwerpunkt bei der Erreichung unserer Ziele. Mit Hilfe von jBPM können diese Prozesse auch grafisch modelliert werden und erhalten ein Stück weit die Einfachheit, die Exceldateien zu Grunde liegt. Diese und andere Prozesssprachen bilden die Schnittstelle zwischen Entwickler und Betriebswirten und spielen in der Industrie eine wichtige Rolle.

Um die Einführung in die Thematik zu erleichtern, wurde das gesamte Projekt in mehrere Teilprojekte zerlegt. Die dort behandelten Problemstellungen - Startpunkt war hier immer eine Exceldatei - stiegen in Arbeitsaufwand und Komplexität. So war das erste Projekt - eine vereinfachte und rudimentäre Lagerverwaltung (siehe Abschnitt 6.1) - lediglich ein erster Einstieg in die verschiedenen, zur Anwendung kommenden Techniken. Die Dauer der Bearbeitung betrug ca. vier Wochen. Für den Rest des ersten Semesters wurde anschließend eine Stundenverwaltung für Angestellte der Technischen Universität Dortmund konzipiert (siehe Abschnitt 6.2). Im zweiten und letzten Semester wurde das umfangreichste Projekt bearbeitet (siehe Abschnitt 6.3) - eine benutzerzentrierte Getränkeverwaltung, die einen Online-Getränkeautomat, eine Lager- und eine Einkaufsverwaltung enthält. Zudem werden statistische Daten aufbereitet und den Benutzern bzw. Verwaltern zur Verfügung gestellt.

Im Laufe des gesamten Projekts war natürlich eine Koordination des Teams und der Arbeit in diesem notwendig. Wir setzten hier eine agile, auf Scrum [Pichler, 2008] basierende, Vorgehensweise ein, die adaptiv und flexibel ist. Agilität bedeutet in diesem Zusammenhang eine schnelle Anpassung an mögliche Komplikationen. Die Software wird - im Gegensatz zu traditionellen Verfahren - Stück für Stück geplant und entwickelt. Dabei steht ein zu jederzeit funktionsfähiges Stück Software im Mittelpunkt. Nähere Informationen dazu findet man in den Kapitel 3.6 und 4.

## 2.3. Aufbau des Endberichts

Zunächst vermittelt dieser Endbericht die theoretischen Grundlagen und Werkzeuge (siehe Kapitel 3), die bei der Entwicklung der serviceorientierten Architekturen eine Rolle gespielt haben. Hier wird das Java Enterprise Edition [Ora, 2010] Umfeld beschrieben, sowie damit verbundene Konzepte und Techniken. Außerdem wird dort die Prozessmanagementplattform *jBPM* [JBoss Community, b] vorgestellt (siehe Abschnitt 3.5) mit der sich Geschäftsprozesse modellieren lassen, und die die Ausführung von Arbeitsabläufen durch mehrere Prozesssprachen unterstützt. Auch die grafischen Manipulationsmöglichkeiten bleiben dort nicht unerwähnt. Desweiteren werden dort die Grundlagen unserer Präsentationsschicht, die auf Apache Tapestry [The Apache Software Foundation, b] basiert, erläutert. Mit diesem Open-Source-Framework (siehe Abschnitt 3.4) für die Programmiersprache Java lassen sich relativ leicht Webanwendungen erstellen. Ebenso findet man dort eine kurze Zusammen-

fassung der Fundamente von Scrum - der agilen Vorgehensweise auf der unser Projektmanagement basierte (siehe Abschnitt 3.6). Im nächsten Kapitel (siehe Kapitel 4) wird das konkrete Vorgehen zur Führung und Verwaltung unserer Projektgruppe erklärt. Hier werden die Unterschiede zu Scrum herausgearbeitet, die sich bei der Organisation unserer PG ergeben haben. Das darauffolgende Kapitel (Kapitel 5) beschreibt das vielgliedrige Schichtenmodell und geht vor allem auf technische Feinheiten in diesem Zusammenhang ein. Die drei verschiedenen Projekte, in die die Projektgruppe zerlegt wurde, werden in Kapitel 6 vorgestellt und in allen Einzelheiten beschrieben. Abschließend betrachten wir das gesamte Projekt noch einmal mit kritischen Augen (Kapitel 7), ehe wir den Bericht mit einer kurzen Zusammenfassung (Kapitel 8) abschließen.



## 3. Grundlagen

Zur Vorbereitung auf die Projekte (s. Abschnitte 6.1 und 6.2) beschäftigten wir uns während der Seminarphase mit Technologien zur Umsetzung mehrschichtiger Software-Anwendungen in der Programmiersprache Java und Projektmanagement für Software-Projekte.

In den nachfolgenden Kapiteln werden zunächst die Technologien beschrieben, die wir in der Projektarbeit verwendet haben.

### 3.1. Java EE

Die *Java Platform, Enterprise Edition (Java EE)* Technologie [Ora, 2010] baut auf der bekannten Java Platform, Standard Edition (Java SE) Technologie auf und dient, wie es der Name bereits vermuten lässt, der Entwicklung von Unternehmenssoftware. Java EE umfasst eine Vielzahl verschiedener Application Programming Interfaces (APIs), die es dem Entwickler ermöglichen sollen, sich ganz auf die eigentliche Funktionalität der Anwendung konzentrieren zu können.

Anwendungen, die nach der Java EE Spezifikation entwickelt wurden, sind verteilte Anwendungen. Das heißt, sie teilen sich in eine Client- und eine Serveranwendung auf, wobei die Serveranwendung, die auch die Anwendungslogik enthält, nochmals in die drei üblichen Schichten (also Präsentationsschicht, Logikschicht und Datenhaltungsschicht) unterteilt ist.

Die *Webschicht* bildet die Schnittstelle zwischen der Clientanwendung und der Geschäftslogik. Sie sorgt dafür, dass die Informationen in die vom Client benötigte Form gebracht werden. In den meisten Fällen bedeutet dies die Erzeugung von HTML-Markup, der dann zum Client geschickt und dort in Form einer Website angezeigt wird.

Die *Geschäftslogikschicht* enthält die eigentliche Funktionalität der Anwendung und nutzt die *Datenbankschicht* zur Persistierung von Daten.

Daraus ergibt sich, dass Java EE Anwendungen aus mehreren Komponenten bestehen (mindestens eine pro Schicht). Jede Komponente nutzt andere von Java EE unterstützte APIs, daher benötigt jede Schicht einen eigenen Container, der eine Laufzeitumgebung für die Komponenten der Schicht bereitstellt. Das gesamte Schichtenmodell, das Java EE zugrunde liegt, ist in Abschnitt 3.2 in Abbildung 3.1 dargestellt.

Um Java EE Anwendungen auszuführen, benötigt man einen speziellen Anwendungsserver, der die Java EE Spezifikation unterstützt.

Die Anwendungen, die im Rahmen dieser Projektgruppe entwickelt werden, bauen auf der Java EE Architektur auf und als Anwendungsserver wird der unter der LGPL-Lizenz frei erhältliche *JBoss Anwendungsserver* [JBoss Enterprise, 2011] in der Version 5.1 eingesetzt.

Nähere Informationen zu dem Schichtenmodell, das unseren Anwendungen zu Grunde liegt, erhält man in Kapitel 5.

## 3.2. EJB

### 3.2.1. Motivation

Die Java EE bietet einige Bausteine, um die Softwareentwicklung schneller und effizienter zu gestalten. Zu diesen Bausteinen zählen unter anderem die Enterprise JavaBeans (EJB) in Version 3.0 [DeMichiel u. Keith, 2006b] und die Java Persistence API (JPA) [DeMichiel u. Keith, 2006c]. Ein großer Vorteil dieser Bausteine ist es, dass so auf einfache Weise verteilte Anwendungen erstellt werden können. Es gibt Frameworks, die auf die vorhandene Technologie aufbauen. Diese helfen unter anderem beim schnellen und komfortablen Erstellen von Sicherheitsmechanismen. Sie können außerdem leicht wiederverwendet werden. Dadurch können wiederkehrende Probleme mit einer einmalig erstellten Lösung abgehandelt werden [DeMichiel u. Keith, 2006b].

Um eine Schnittstelle zu gewöhnlichen Java SE Anwendungen zu schaffen, können EJB auch aus der Java SE heraus benutzt werden. Dabei wurde ein besonderes Augenmerk auf die Einfachheit der Entwicklung gelegt. In mehrschichtigen Anwendungen vereinfachen sie die Kommunikation mit der Datenbank. Das zeigt sich zum Beispiel in Abbildung 3.1. Es handelt sich dabei um eine Applikation, die mit einer Datenbank kommuniziert. Die Kommunikation erfolgt allerdings nicht direkt aus der Anwendung heraus, sondern ist durch EJB gekapselt. Daher kann der Anwendungsentwickler sein Anwendungsprogramm schreiben, ohne Gefahr zu laufen, verschiedene Dinge wie Anwendungslogik und Datenbankenverbindungsaufbau miteinander zu vermischen. Es ist auch kein detailliertes Wissen über Datenbanken und deren Besonderheiten beim Auswählen und Einfügen von Daten nötig.

Die Kommunikation mit der Datenbank kann dabei mit der JPA abgekapselt werden. Sie ist Bestandteil der Java EE. Aus Datensätzen können so Objekte generiert werden. Bei diesen Objekten handelt es sich um normale Java Objekte. Es wurde auf eine einfache Handhabung der API Wert gelegt. Sie schließt die Lücke zwischen objektorientierter Programmierung und relationalen Datenbanken. Dabei wurde, wann immer es ging, auf vorhandene Standards zurückgegriffen. Das zeigt sich zum Beispiel an der Verwendung von Extensible Markup Language (XML). Es wurde eine eigene Anfragesprache, die Java Persistence Query Language, geschaffen, die eine Abstraktion zu den unterschiedlichen Abfragen der verschiedenen Datenbanken schafft [DeMichiel u. Keith, 2006c] [Ora, 2010].



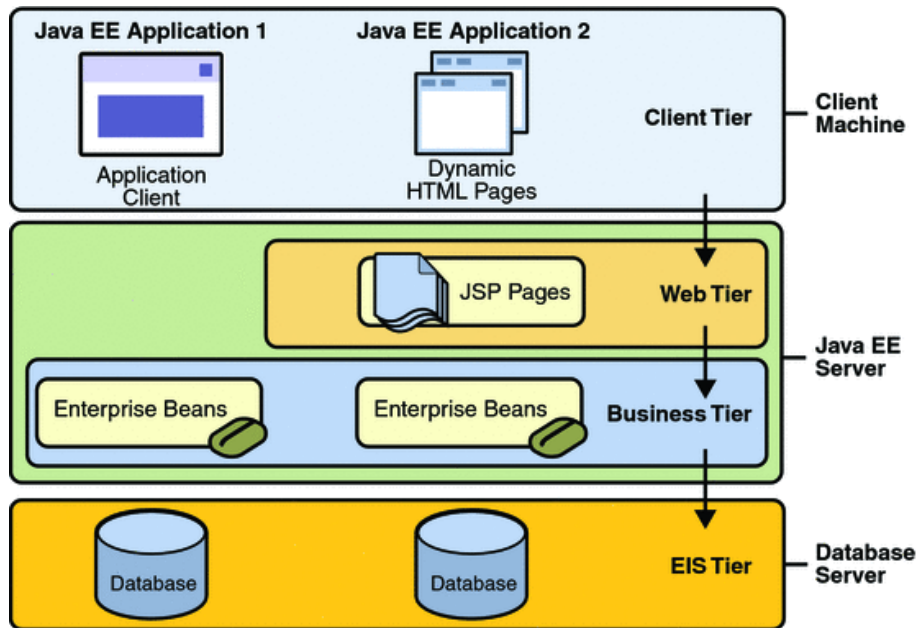


Abbildung 3.1.: EJB in mehrschichtigen Anwendungen Ora [2010]

### 3.2.2. Session Beans und Message-Driven Beans

Die typische *Session Bean* stellt eine Verbindung für einen einzelnen Client her. Sie kann Transaktion unterstützen. Dabei greift sie, wie in Abbildung 3.1 dargestellt, auf eine Datenbank zu. Diese Datenbank kann abgekapselt und auf einem entfernten System laufen. Die Daten der Membervariablen der Bean müssen nicht zwingend auch alle Daten direkt aus der Datenbank repräsentieren. Ein Zugriff auf diese kann erfolgen und das Aktualisieren der Daten ist auch möglich. Die Session Beans leben im Vergleich zu anderen Beans nur sehr kurz. Sollte der EJB-Container abstürzen, so wird das Session Bean zerstört. Der Client muss dann ein neues Objekt anfordern, wodurch die Sicherheit erhöht wird. Das klassische Einsatzszenario für die Session Beans ist die Kommunikation mit der Datenbank. Diese werden unter anderem bei Kontotransaktionen oder in Onlinekaufhäusern benötigt.

Der typische EJB-Container unterstützt das performante Erstellen von vielen Session Beans zur selben Zeit und skaliert dabei auch gut. Es gibt zwei Arten von Session Beans: die zustandsabhängigen und die zustandslosen Sessions. Beide unterscheiden sich etwas in ihrem Anwendungsbereich und in ihrer API [DeMichiel u. Keith, 2006b].

Eine *Stateless Session Bean* hat keinen gemeinsamen Zustand mit dem Client. Wenn der Client eine Methode einer zustandslosen Session Bean aufruft, dann wird eine neue Bean generiert und verwendet. Es sind keine Informationen aus vorherigen Sitzungen vorhanden. Wenn eine Methode verlassen wird, werden auch alle vorhandenen Informationen verworfen. Lediglich über die Parameterliste können Informationen von dem Client an die Session Bean übergeben werden. Außer beim Me-

thodenaufruf sind alle Instanzen der Bean für alle Clients gleich. Der EJB-Container weist einem Client dann eine bestimmte Instanz zu, auf welcher dieser dann Aufrufe starten kann.

Die Skalierbarkeit des Systems ist sehr gut, da der EJB-Container immer ausreichend Instanzen bereithalten kann. Zu den einzelnen Instanzen müssen keine Zustände gespeichert werden. Wenn ein Client mit einer Instanz fertig ist, kann sie direkt für einen anderen Client wiederverwendet werden.

Die zustandslosen Session Beans eignen sich hervorragend zur Implementierung von Webservices, da diese auch zustandslos sind [Ora, 2010].

Hingegen bei den *Stateful Session Beans* wird ein gemeinsamer Zustand in den Instanzvariablen der Bean gespeichert. Diese Variablen präsentieren einen Zustand, in dem sich genau eine Bean befindet. Wenn der Client mit der Bean interagiert, ändern sich diese Variablen und der Zustand ändert sich. Der Zustand wird für die ganze Sitzung des Clients beibehalten. Es muss also für jeden Client eine eigene Bean vorhanden sein. Falls der Client die Bean löscht oder beendet, wird sie verworfen, da keine Notwendigkeit besteht, sie weiterhin zu speichern.

Die Implementierung einer zustandsabhängigen Session Bean verläuft analog zur Implementierung einer Stateless Session Bean. Der größte Unterschied ist die Möglichkeit, Membervariablen des Objekts in einzelnen Methoden zu verwenden [Ora, 2010].

Ein weiterer Typ von Beans ist die *Message-Driven Bean*. Sie verhält sich in einigen Punkten ähnlich wie eine Stateless Session Bean. Sie ist zustandslos und kann transaktionssicher sein. Sie kann beispielsweise Daten mit einer Datenbank synchronisieren und ist relativ kurzlebig. Sollte der EJB-Container abstürzen, wird sie entfernt. Es muss dann eine neue Instanz der Bean erstellt werden. Der Unterschied zu Session Beans liegt darin, dass sie nach dem Empfangen einer Nachricht vom Client auf dem Server ausgeführt wird. Die Ausführung erfolgt asynchron. In einem typischen EJB-Container können sehr viele Message-Driven Beans parallel existieren. Dadurch muss der Container auch für viele Message-Driven Beans gut skalieren [DeMichiel u. Keith, 2006b]. Sie eignen sich hervorragend zur Implementierung von Webservices. Dazu gibt es detailliertere Informationen in Kapitel 3.3.

### 3.2.3. Entity Beans

Für gewöhnlich sind Entity Beans Bestandteil des Domänenmodells. Sie bieten eine Objektsicht auf die Informationen in der Datenbank. Daher existieren diese Objekte unter Umständen sehr lange - nämlich solange, wie sich die Informationen in der Datenbank befinden. Im Gegensatz zu den bislang vorgestellten Beans überlebt eine Entity Bean den Absturz des EJB-Containers, da ihr Primärschlüssel erhalten bleibt. Sollte der Container abstürzen, bevor eine Transaktion komplett ausgeführt wurde, so wird der Zustand vor der Transaktion beim nächsten Zugriff auf die Bean wiederhergestellt. Der EJB-Container kann normalerweise eine große Anzahl an Entity Beans auch konkurrierend zueinander verwalten [DeMichiel u. Keith, 2006b].

### 3.2.4. Transaktionen

Soll in einer Software ein Prozess abgebildet werden, so kann dieser aus mehreren Schritten bestehen. Oft ist es dann nötig, dass diese Schritte atomar durchgeführt werden. Dies bedeutet, dass entweder alle Schritte erfolgreich durchgeführt oder keiner. Falls der Prozess auf unterschiedlichen Datensätzen zugreift, kann nur so die Integrität der Daten sichergestellt werden. Daher werden mehrere Operationen zu einer unteilbaren Transaktion zusammengefasst. Wenn eine Transaktion abgeschlossen werden soll, muss ein Commit erfolgen. Sollen die Änderungen verworfen werden, so kann dieses per Rollback geschehen. Das kann nützlich sein, wenn es bei einer der Operationen in der Transaktion zu einem Fehler oder einer Ausnahme kommt. Es ist dann sichergestellt, dass auch die zuvor vorgenommenen Veränderungen nicht durchgeführt werden. Das ist zum Beispiel von Bedeutung, wenn ein Artikel in einem Shop gekauft wird und anschließend festgestellt wird, dass der Artikel nicht geliefert werden kann oder nicht in ausreichender Stückzahl vorrätig ist. Dann soll selbstverständlich auch keine Abbuchung vom Benutzerkonto erfolgen. Ein solcher Einkaufsprozess ist ein typischer Anwendungsfall für Transaktionen.

In EJB gibt es zwei unterschiedliche Arten von Transaktionen. Die einen werden vom Container und die anderen durch die Bean selbst gesteuert. Beide haben ihre Vor- und Nachteile. Sie definieren unterschiedliche Start und Endpunkte der Transaktionen [DeMichiel u. Keith, 2006c] [DeMichiel u. Keith, 2006b].

In einer EJB, die in einer vom Container verwalteten Umgebung (*Container Managed Transaction*) läuft, gibt es für die Transaktionen keinen Start oder Endpunkt. Dieses wird alles automatisch vom Container verwaltet. Es kann sich dabei um Session oder auch um Message-Driven Beans handeln. Die Entwicklung wird so dramatisch vereinfacht, da sich der Entwickler keine Gedanken über die Transaktionen, ihren Start und ihr Ende machen muss. Ist kein anderes Verhalten an die Bean annotiert, so ist dieses auch das Standardverhalten.

Für gewöhnlich startet der Container die Transaktion bevor eine Methode der Bean aufgerufen wird. Er beendet die Transaktion bevor die Methode verlassen wird. Somit ist der Methodenaufruf komplett in eine eigene Transaktion gekapselt. Mehrfache Transaktionen und verschachtelte Transaktionen sind bei diesem Modell nicht erlaubt. Welche Methoden von der Transaktion betroffen sind, lässt sich mit einer Annotation markieren. Es müssen nicht alle Methoden Transaktionen verwenden [Ora, 2010].

In den beangesteuerten Transaktionen (*Bean Managed Transaction*) muss die Transaktion vom Programm explizit gestartet und beendet werden. Dadurch wird der Bereich, der in eine Transaktion hineinfällt, fest bestimmt. Die Transaktionen können beliebig gestartet und beendet werden. Es können mehrere Transaktionen ineinander und umeinander geschachtelt sein. Ein Methodenaufruf ist nicht auf eine Transaktion beschränkt. Die Transaktionen können entweder durch Java Database Connectivities (JDBC) (siehe [Oracle, b]) oder die Java Transaction API (JTA) (siehe [Ora, 2010]) umgesetzt werden. Um JTA Transaktionen benutzen zu können, muss das `javax.transaction.UserTransaction` Interface implementiert werden.

Auf dem Objekt können dann die `begin`, `commit` und `rollback` Methoden ausgeführt werden [Ora, 2010].

### 3.2.5. JPA

Die Java Persistence API (JPA) (siehe [Biswas u. Ort, 2006]) ist nicht nur Bestandteil von Java EE sondern kann auch außerhalb der Java EE Umgebung eingesetzt werden. Verschiedenste Applicationen wie Desktopclientsoftware oder Webanwendungen können die JPA verwenden. Es gibt verschiedene Implementierungen der API. Sie ist dafür geschaffen worden, um ein Mapping von Objekten auf relationale Datenbanken zu vollziehen. Dieser Vorgang wird Objekt/relationales Mapping (ORM) genannt. Der Benutzer kann dabei direkt auf den Objekten, z.B. den Entity Beans, arbeiten. Anfragen an die Datenbank mit SQL-Abfragen werden automatisch gesendet und müssen nicht manuell erledigt werden.

Die Abbildung der Daten aus der Datenbank erfolgt dabei über Metadaten. Die JPA definiert diese Metadaten über Annotationen in den Java-Klassen. Alternativ können die Metadaten auch in externen XML Daten abgelegt werden. Eine Kombination aus beidem ist auch möglich, wobei die Daten in der XML-Datei die aus den Annotationen überschreiben. Durch die Metainformationen in den Annotationen wird dann die richtige Datenbankoperation und -verbindung ausgewählt [DeMichiel u. Keith, 2006c].

Zur Verwaltung der Verbindungen und Anfragen an die Datenbank gibt es einen *Entity Manager*. Dieser verweist auf den so genannten *Persistenzkontext*. Dieser verwaltet die Datenhaltung der gesamten EJB Anwendung. Dabei handelt es sich um die Umgebung in der alle Datenbankoperationen ausgeführt werden.

Es gibt zum einen die *Container-Managed Entity Beans*. Bei ihnen wird der Entity Manager vom EJB-Container bereitgestellt (vergleiche dazu [Ora, 2010]). Der Entity Manager verteilt die nötigen Informationen an alle beteiligten Komponenten der Anwendung. Die Transaktionen führen Aufrufe zwischen den einzelnen Komponenten der Anwendung aus. Sobald eine Transaktion abgeschlossen werden soll brauchen die Komponenten, einen eindeutigen Persistenzkontext. Diesen können sie durch den Entity Manager bekommen. Er ist für die Verwaltung der Transaktionen zuständig.

Eine andere Möglichkeit einen eindeutigen Persistenzkontext zu erhalten ist es, dass sich die Anwendung selbst darum kümmert. Bei den *Application-Managed Entity Beans* wird der Entity Manager von der Anwendung selbst und nicht vom EJB-Container erzeugt. Daher wird der Lebenszyklus durch die Anwendung bestimmt. Sie erstellt den Manager und zerstört ihn auch wieder. Der Persistenzkontext wird nicht an alle Komponenten der Anwendung automatisch verschickt. Immer wenn die Anwendung einen Persistenzkontext benötigt, muss dieser von der Applikation selbst bereitgestellt und gefunden werden [Ora, 2010].

## 3.3. Webservices

Im Zuge der immer größeren Verteilung und Vernetzung von Software ist der Bedarf von standardisierten Kommunikationskanälen immer größer geworden. Zu diesem Zweck sind die Webservices entstanden. Sie bieten Schnittstellen, die auf Funktionen von Programmen zugreifen. Durch eine standardisierte Kommunikation kann das Programm von anderen Programmen gesteuert werden. Die Kommunikation ist dabei abgelöst von proprietären Protokollen und kann frei verwendet und wiederverwendet werden.

Die verschiedenen Beschreibungen für die Kommunikation werden dabei vom *World Wide Web Consortium (W3C)* [w3c, 2011] vereinbart. Durch die weite Akzeptanz des Standards gibt es viele Implementierungen, die sich an diesen Vorgaben orientieren und somit die Chance bieten untereinander kompatibel zu sein.

Der Oberbegriff Webservices beinhaltet viele Technologien, die für unterschiedliche Problemstellungen Lösungen bereitstellen. Die Seiten des W3C [Lafon, 2002] beinhalten ausführliche Dokumentationen zu Optimierungen, Sicherheitsaspekten, Nachrichtentransport und weiterführenden Thematiken.

### 3.3.1. Webservice-Technologie

Das Konstrukt der Webservices hat drei zentrale Aspekte: Es gibt einen Konsumenten, der auf den Service zugreift, es gibt eine Beschreibung des Services und es gibt den eigentlichen Dienst selbst (s. Abbildung 3.2).

Die Beschreibung der Schnittstelle des Dienstes erfolgt per *Web Services Description Language (WSDL)* [Christensen u. a., 2001]. In dieser XML-Datei [Newcomer, 2002] werden alle Objekte und Funktionen des Dienstes beschrieben und bekannt gemacht. Der Konsument kann auf die beschriebenen Objekte mit den beschriebenen Funktionen zugreifen und diese bearbeiten.

Soll automatisch auf einen Server mit mehreren verschiedenen Diensten zugegriffen werden, so kann es eine *Universal Description, Discovery and Integration (UDDI)* [Newcomer, 2002] Komponente geben. Diese verschickt die unterschiedlichen Beschreibungen der einzelnen Webservices an die anfragenden Konsumenten.

Die Kommunikation des eigentlichen Dienstes mit dem Konsumenten erfolgt auf die in der WSDL festgelegte Weise. Der Dienst kann sich beim UDDI registrieren (siehe dazu Punkt 1 in Abbildung 3.2). Sofern dies geschehen ist, kann der Konsument die WSDL zu dem entsprechenden Service beim UDDI Dienst anfragen (siehe Punkt 2 und 3 in Abbildung 3.2). Nachdem der Konsument die WSDL zur Kommunikation erhalten hat, kann er mit dem Dienst kommunizieren (Abbildung 3.2, Punkt 4 und 5). Sofern der Konsument die WSDL auf anderem Weg erhält, kann die Kommunikation auch direkt gestartet werden. Die Punkte 1 bis 3 in Abbildung 3.2 sind also optional und nur beim Vorhandensein von UDDI erforderlich.

Die Übertragung der Informationen kann über verschiedene Wege erfolgen. Als Basis zur Übertragung von Daten dient dem Webservice das *Hypertext Transport Protokoll (HTTP)*.

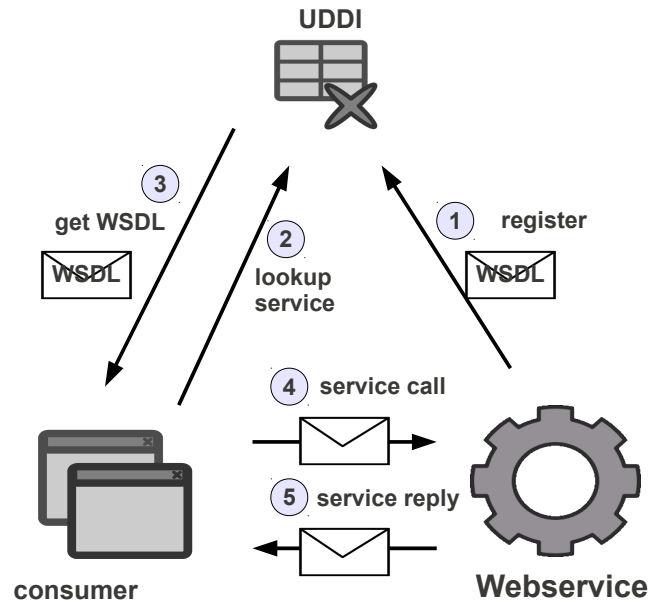


Abbildung 3.2.: Service Struktur

Es wird zwischen *Representational State Transfer (REST)*ful Webservice [Rodriguez, 2006] und *Simple Object Access Protocol (SOAP)* [Newcomer, 2002] unterschieden. Beide Technologien haben unterschiedliche Schwerpunkte hinsichtlich der übertragenen Daten und der damit verbundenen Verarbeitung in der Anwendung.

Bei SOAP werden die Daten in einen XML-Stream verwandelt. Alle Informationen werden damit als Zeichenkette übertragen. Die übertragene Nachricht ist somit mit allen Vor- und Nachteilen von XML behaftet. Die Nachricht kann mittels *Extensible Stylesheet Language (XSL)* validiert werden. Der Webservice wird über eine WSDL-Datei beschrieben. Das Ansteuern von Objekten kann so sehr gezielt und präzise erfolgen. Nachteile liegen im Bereich von binären Dateiübertragungen, wie sie für Multimedia und Bilder üblich sind. Hier muss der Binärstrom erst in eine Zeichenkette umgewandelt werden und kann anschließend in einer XML Nachricht verpackt werden. Die Struktur der SOAP-Nachricht erzeugt den für XML üblichen zusätzlichen Overhead. Dafür ist es sprach- und plattformübergreifend einsetzbar.

Bei RESTful Webservices [Rodriguez, 2006] ist das zu übertragende Format von dem aktuellen Zustand, in dem sich der Webservice befindet, abhängig. Das bedeutet konkret, dass es keine einheitliche Kommunikation per XML oder ähnlichem gibt. Die Daten werden einfach wie bei HTTP üblich übertragen. Bilder und Multimediaformate können so einfach übertragen werden. Der für XML übliche Overhead entfällt dabei mitsamt aller Vorzüge und Seiteneffekte. Die Steuerung erfolgt über wenige Befehle zum Übertragen von Informationen. Diese sind an die Kommandos von HTTP angelehnt [Rodriguez, 2006].

### 3.3.2. JAX-WS Projekt

Eine einfache Möglichkeit Webservices im Javaumfeld zu erstellen bietet das *Java API for XML - Web Services (JAX-WS)* Projekt [Oracle, 2011] an. Es ist seit Java 5 Bestandteil von Java EE. Webservices können mit Hilfe von JAX-WS automatisch generiert werden. Zur Steuerung kommen Annotationen zum Einsatz. Die Generierung von WSDL-Dateien kann durch Tools automatisch erfolgen, genau wie die Bereitstellung auf dem Webserver. Der Programmierer muss sich ausschließlich um die Funktionalität kümmern.

Die Verwendung von Plain Old Java Objects (POJOs) ist möglich. Es müssen lediglich einige Annotationen zur Steuerung der Veröffentlichung hinzugefügt werden. Die Entwicklung kann durch diesen Automatismus in zwei Richtungen erfolgen: Es können vorhandene POJOs in Webservices verwandelt werden, aber es können auch Objekte aus vorhandenen WSDL-Dateien generiert werden. Die Funktionalität muss dabei selbstverständlich der Programmierer hinzufügen.

JAX-WS unterstützt sowohl SOAP als auch RESTful Webservices [Tyagi, 2006]. Die Steuerung um welche Art von Webservice es sich handeln soll erfolgt dabei - wie die gesamte Steuerung - über Annotations.

## 3.4. Tapestry

Während der Seminarphase der Projektgruppe beschäftigten sich die Teilnehmer mit den beiden OpenSource-Frameworks Wicket [The Apache Software Foundation, c] und Tapestry [The Apache Software Foundation, b] zur Erstellung von Webanwendungen in der Programmiersprache Java. Beide Frameworks werden von der Apache Software Foundation entwickelt und arbeiten nach dem Model-View-Control (MVC)-Prinzip (s. Abschnitt 3.4.2). Dabei verfolgen sie jedoch teilweise sehr unterschiedliche Ansätze.

Nach einem Vergleich beider Frameworks ist die Wahl letztendlich auf Tapestry gefallen, da die Teilnehmer der Projektgruppe den Eindruck hatten, dass sie sich in Tapestry schneller einarbeiten können und der Fokus der Projektarbeit nicht in der GUI, sondern der Prozessebene angesiedelt ist.

Die folgenden Abschnitte geben somit einen Überblick über den Aufbau einer Tapestry-Anwendung.

### 3.4.1. Konvention über Konfiguration

Tapestry setzt stark auf das Prinzip der *Konvention über Konfiguration*. Dabei werden folgende Konventionen unterstützt:

- Paketbezeichner  
In der *web.xml* wird ein Basispaket für die Anwendung spezifiziert. Ausgehend von diesem Basispaket erwartet Tapestry Java-Klassen für Seiten (s. Abschnitt

3.4.3) im Unterpaket *pages* und Komponenten (s. Abschnitt 3.4.5) im Unterpaket *components*. Zusätzlich werden per Konvention ein Unterpaket *services* für Services aller Art und ein Unterpaket *model* für Models erwartet.

- **Verzeichnislayout**  
Im Webverzeichnis werden die Seitentemplates (s. Abschnitt 3.4.3) erwartet. Templates für Komponenten (s. Abschnitt 3.4.5) hingegen werden im Resource-Verzeichnis entsprechend dem Paketnamen erwartet.
- **Methoden-Benennung**  
Bei der Behandlung von Ereignissen (s. Abschnitt 3.4.6) können Methoden entsprechend dem zu behandelnden Ereignis benannt werden, wodurch die Methoden automatisch beim Auftreten eines Ereignisses aufgerufen werden können. Somit kann auf einfache Weise durch Einhaltung der Konvention auf Ereignisse innerhalb einer Komponente oder Seite reagiert werden.

Diese Konventionen sind gut durchdacht und helfen dabei ein Tapestry-Projekt gemäß dem MVC-Entwurfsmuster (s. Abschnitt 3.4.2) umzusetzen und die einzelnen Bereiche nicht zu mischen. Natürlich können Konventionen dabei nur helfen und eine Mischung nicht verhindern.

### 3.4.2. MVC-Entwurfsmuster (MVC-Pattern)

Die Unterteilung in Seiten, Templates und Modelle in Tapestry-Anwendungen dient der Software-Entwicklung nach dem MVC-Pattern (siehe Abbildung 3.3). Dabei handelt es sich um ein Architekturmuster, welches ein Programm in die drei Schichten Model, View und Controller einteilt. Hierbei wird versucht die Abhängigkeiten zwischen den Schichten so gering wie möglich zu halten, um die jeweiligen Schichten einfach modifizieren oder austauschen zu können.

Jede dieser drei Schichten besitzt eigene Aufgaben:

- **das Model** stellt hierbei die Businesslogik dar und stellt Methoden bereit, um Daten abrufen und manipulieren zu können, wobei das View über vorgenommene Änderungen über Events informiert wird.
- **das View** stellt die Daten des Models dar und leitet Benutzeraktionen (über Events) an einen Controller weiter.
- **der Controller** nimmt Benutzeranfragen entgegen, manipuliert gegebenenfalls das Model und wählt eine View zur Ansicht für den Benutzer aus.

Lose Bindungen, in Form von Events, existieren zum einen vom Model zur View, um dieses über Änderungen im Datenbestand zu informieren, sowie vom View zum Controller, um Benutzeraktionen auszulösen.

Die Umsetzung des MVC-Patterns in Tapestry unterscheidet sich dabei leicht vom klassischen Ansatz. Das View fragt das Model nicht direkt ab, stattdessen fragt der



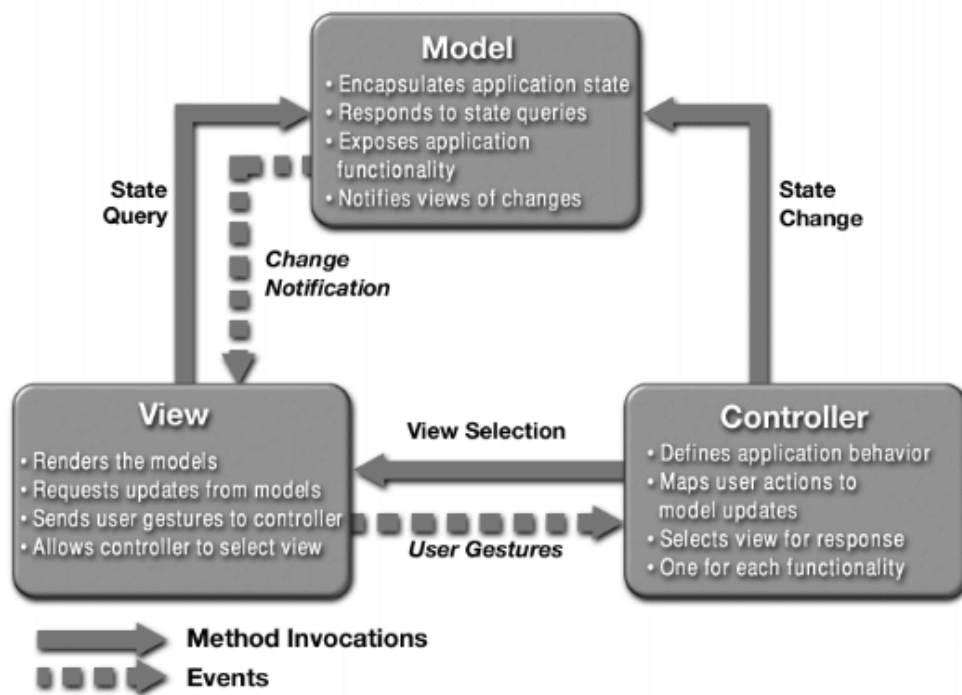


Abbildung 3.3.: Schichten einer MVC-Anwendung [Sun-Microsystems]

Controller das Model ab und reicht die Daten an das View weiter. Die lose Bindung aus dem Model heraus entfällt vollständig, da es sich bei Tapestry-Anwendungen um Web-Anwendungen handelt und Änderungen im Model somit stets nur über einen erneuten Request im Browser dargestellt werden können.

### 3.4.3. Seiten (Pages)

Eine Tapestry-Anwendung ist zunächst in unterschiedliche Seiten aufgeteilt, welche aus einer Java-Klasse und einem Template bestehen. Die Java-Klasse ist ein POJO - muss also nicht von einer bestimmten Basisklasse erben oder bestimmte Interfaces implementieren.

Zu jeder Seite gehört ein Template, welches die View für die Page bestimmt. Das Template ist ein wohlgeformtes XML-Dokument und kann neben einfachem XHTML-Code zusätzlich Elemente aus der *Tapestry Markup Language* (s. Abschnitt 3.4.4) enthalten.

Wird eine Anfrage von Tapestry entgegen genommen, so wird ein Objekt von der entsprechenden Page-Klasse erstellt und für den Zugriff durch das Template beim Parsen zugänglich gemacht. Per Konvention sind die Page-Klassen der Anwendung im Paket *<Basispaket>.pages* und Templates mit der Endung *tml* im Verzeichnis *src/main/webapp* der Anwendung zu speichern, wobei der Dateiname jeweils dem Namen der Klasse entspricht.

Beispiele:

URL	Java-Klasse	Java-Datei	Template
/	Index	Index.java	Index.tml
/index	Index	Index.java	Index.tml
/products/books	products.Books	products/Books.java	products/Books.tml

Eine weiterführende Konfiguration ist nicht notwendig. Insbesondere ist es nicht notwendig, eine Zuordnung von Java-Klasse zu einem Template für eine Page vorzunehmen, da dies durch die Einhaltung der Namenskonvention automatisch von Tapestry vorgenommen werden kann.

### 3.4.4. Tapestry Markup Language

Innerhalb der Templates bildet die *Tapestry Markup Language* die Schnittstelle zum Java-Code, mit dessen Hilfe dynamische Inhalte in das Template eingefügt werden können.

Mittels sogenannter Expansionen können Methoden von Java-Objekten aufgerufen werden, wobei der Rückgabewert dann an die entsprechende Stelle im Template eingefügt wird. Eine Expansion hat die Form `${foo.bar}`. Dabei werden die Elemente `foo` und `bar` auf die Methoden `getFoo()` bzw. `getBar()` abgebildet. Die Expansion `${foo.bar}` entspricht also in Java einem verketteten Aufruf von `getFoo().getBar()`.

Zusätzlich zu Expansionen bietet die *Tapestry Markup Language* noch eigene Elemente, wie z.B. `<t:pagelink ...>` und `<t:actionlink ...>`, welche dazu genutzt werden können um auf eine andere Seite zu verlinken oder eine Aktion bzw. ein Event innerhalb einer Seite auszulösen, worauf dann innerhalb der Java-Klasse reagiert werden kann (s. Abschnitt 3.4.6). Elemente werden dabei auf Komponenten (s. Abschnitt 3.4.5) abgebildet und nicht auf einzelne Methoden.

Für detailliertere Informationen wird auf [The Apache Software Foundation, b] verwiesen.

### 3.4.5. Komponenten

Seiten für sich genommen wären nur für recht kleine Projekte ausreichend und würden schnell unübersichtlich, da eine Seite sämtliche Funktionalitäten der Bereiche, die in ihr dargestellt werden, zur Verfügung stellen muss.

Nun gibt es bei einer Web-Anwendung aber bestimmte Bereiche, die eine logische Einheit bilden und eventuell auch mehrfach verwendet werden können. Um diese nicht in jeder benötigten Seite implementieren zu müssen, bietet Tapestry neben Seiten auch Komponenten an, die wie Bausteine im Template einer Page eingefügt werden können. Komponenten sind ähnlich aufgebaut wie Seiten. Es gibt also eine Java-Klasse, in welcher die serverseitige Logik programmiert wird und zusätzliche Templates, um die Ansichten zu generieren. Mit Hilfe von Komponenten können wiederkehrende Problemstellungen zentral gelöst und mehrfach verwendet werden.

Gute Beispiele für wiederverwendbare Komponenten stellen die mitgelieferten Standard-Komponenten von Tapestry dar. Beispielsweise bietet Tapestry eine Komponente *beaneditform*, welche aus einem übergebenen Objekt automatisch ein Eingabeformular generiert und eine Komponente *grid*, welche eine Auflistung aus einer Liste von Objekten generieren kann. Mittels der Komponenten *pagelink* und *actionlink* lassen sich Links generieren, die einen Seitenwechsel hervorrufen oder Events innerhalb der Komponente (bzw. Seite) auslösen.

### 3.4.6. Event-Behandlung

Wie im vorherigen Abschnitt 3.4.5 beschrieben, gibt es Komponenten, die Events auslösen können. Ein Event besteht aus einem Typen und einer optionalen ID, welche den Ursprung des Events angeben kann. Auf diese Art kann unterschiedlich auf Events reagiert werden.

Um nun auf ein Event zu reagieren, kann in einer Seite oder Komponente eine Methode *onEventType[FromComponentId]* implementiert werden, welche dann automatisch von Tapestry aufgerufen wird. Wird die Komponente *actionlink* verwendet, so ist der Typ des Events *action* und eine Methode *onAction* würde auf sämtliche Events dieses Typs reagieren. Mehrere Events gleichen Typs werden anhand von IDs unterschieden. So kann der Komponente *actionlink* ein Parameter *id* übergeben werden, welcher der Identifikation dient. In diesem Fall ist die Methode dann zum Beispiel als *onActionFromEdit* zu benennen, wenn der Typ des Events *action* und die ID *edit* beträgt.

Tapestry sucht dabei bei jedem auftretenden Event zunächst innerhalb der eigenen Komponente nach einer passenden Methode, welche das Event behandeln könnte. Falls innerhalb der eigenen Komponente keine passende Methode gefunden wurde, so wird in der Komponenten-Hierarchie solange nach oben gestiegen bis auf oberster Ebene innerhalb der Seite selbst eine passende Methode gesucht wird. Sollte auch in der Seite keine passende Methode gefunden werden, so wird das Event abgebrochen und nichts unternommen.

Für den Fall, dass einem die Namenskonvention nicht zusagt, bietet Tapestry die Möglichkeit, die Methoden frei zu benennen und per Annotations das Event und die Komponente anzugeben (s. Listing 3.1).

Listing 3.1: Freier Methoden-Name für Events

```
public class Index {
    // ...
    @OnEvent(value="action", component="link")
    void foo() {
        System.out.println("OnActionFromLink");
    }
    // ...
}
```

Wird ein Event zum Beispiel durch einen Link ausgelöst, so wird zunächst die Seite mit sämtlichen Komponenten geladen, welche das Event ausgelöst hat und - falls nicht anders angegeben - gerendert.

Für den Fall, dass ein Seitenwechsel stattfinden soll, bietet Tapestry die einfache Möglichkeit, ein Objekt der anzuzeigenden Page als Rückgabewert einer Eventfunktion zu verwenden (s. Listing 3.2):

Listing 3.2: Rückgabe einer Page führt zu einem Seitenwechsel

```
public class Index {
    @InjectPage
    NewPage newPage;

    NewPage foo () {
        // ...
        return newPage;
    }
}
```

Soll die gleiche Seite gerendert werden, also kein Seitenwechsel stattfinden, so kann entweder *void* als Rückgabotyp der Methode verwendet (s. Listing 3.1) oder *null* zurückgegeben werden.

In Listing 3.2 ist weiterhin zu sehen, dass der Programmierer sich selbst nicht um die Erzeugung des Seiten-Objektes kümmern muss. Dies kann von Tapestry automatisch injiziert werden. Der Programmierer muss lediglich ein Attribut vom Typen des Seiten-Objektes anlegen und mit Hilfe einer Annotation angeben, dass dieses Objekt injiziert werden soll. Alles weitere wird von Tapestry übernommen.

### 3.4.7. Nachrichten-Kataloge

Zur Auslagerung von Zeichenketten und zur Internationalisierung werden in Tapestry sogenannte Nachrichten-Kataloge verwendet. Dies sind gewöhnliche Java-Properties-Dateien, also Textdateien, in denen jede Zeile einen Schlüssel und einen Wert enthält, welche durch ein “=” getrennt werden (s. Listing 3.3).

Listing 3.3: Beispiel-Nachrichten-Katalog

```
title=Newsletter-Anmeldungen
subtitle=Diese Anwendung dient der Verwaltung von \
    Newsletter-Anmeldungen
createNewSubscriber=Neuen Teilnehmer anlegen
delete=Loeschen
```

Innerhalb eines Templates kann mit Hilfe der Expansion  $\${message:key}$  (s. Abschnitt 3.4.4) auf eine Zeichenkette eines Nachrichten-Katalogs zugegriffen werden. Neben einem globalen Nachrichten-Katalog *app.properties*, welcher sich im WEB-INF-Verzeichnis der Anwendung befindet, kann jede Komponente oder Seite eigene zusätzliche Kataloge verwenden.

Nachrichten-Kataloge für Seiten (bzw. Komponenten) werden per Konvention im Verzeichnis *src/main/resources/<paket>* gespeichert und tragen den Namen *<page>.properties* (bzw. *<component>.properties*). Neben einem Standard-Nachrichten-Katalog, der als Fallback dient, können noch sprachspezifische Nachrichten-Kataloge verwendet werden. In diesem Fall ist dem Dateinamen noch ein *\_<ll>[\_<CC>]* anzuhängen. Hierbei steht *<ll>* für den Sprach- und *<CC>* für den Ländercode, welcher jedoch optional ist.

### 3.4.8. Validierung

Zur Überprüfung von Eingabedaten bietet Tapestry Validierer an, welche direkt innerhalb der Seiten- oder Komponenten-Klassen mittels einer Annotation *@Validate(...)* an den Attributen angegeben werden können. Tapestry bietet folgende vordefinierte Validierer an:

email	Der Wert muss eine gültige E-Mail-Adresse repräsentieren
max	Der Wert muss eine Zahl sein und darf den Wert von max nicht übersteigen
maxLength	Der Wert muss eine Zeichenkette sein und darf die angegebene Länge nicht übersteigen
min	analog zu max
minLength	analog zu maxLength
regex	Der Wert muss auf den angegebenen regulären Ausdruck passen
required	Es muss ein Wert angegeben sein

Zusätzlich zur Angabe als Annotation im Java-Quellcode, können Validierer auch im Template als Attribut eines Formular-Elementes angegeben werden, s. Listing 3.4.

Listing 3.4: Validierer im Template

```
<t:textfield ... validate="required ,email" />
```

Sollten die zur Verfügung stehenden Validierer nicht ausreichen, steht für einfache Fälle das Event *ValidateForm* zur Verfügung, welches in einer Methode *onValidateForm* behandelt werden kann. Dort bietet Tapestry die Möglichkeit, Eingaben differenzierter zu überprüfen und gegebenenfalls eine Fehlermeldung zu produzieren.

Müssen Überprüfungen vorgenommen werden, die an mehreren Stellen im Programm eingesetzt werden, bietet Tapestry überdies noch an eigene Validierer zu definieren, die dann wiederum einfach als Annotation im Java-Quellcode oder Attribut im Template angegeben werden können.

### 3.4.9. Persistierung

Da es sich bei HTTP um ein statusloses Protokoll handelt, was bedeutet, dass der Status der Anwendung nach jedem Request "vergessen" wird, ergibt sich die Anforderung

derung, Daten über mehrere Requests hinweg zu speichern. Tapestry bietet hierzu die Möglichkeit, Daten für eine gewisse Dauer innerhalb der Session oder permanent in einer Datenbank zu persistieren. Hierzu implementiert Tapestry die JPA (s. Abschnitt 3.2.5), welche über Annotations sehr einfach zu verwenden ist. Innerhalb der Session können Attribute mittels *@Persist* gespeichert werden.

Sollen Daten nur bis zum nächsten Request in der Session gehalten werden, so steht die Annotation *@Persist("flash")* zur Verfügung, welche den Wert nur im folgenden Request wiederherstellt und anschließend vergisst.

Zusätzlich bietet Tapestry die einfache Möglichkeit Daten mittels Hibernate [JBoss Community, a] in einer Datenbank zu organisieren. Die Entitäten werden per Konvention im Unterpaket *model* erwartet. Dies sei hier jedoch nur am Rande erwähnt, da im Rahmen dieser Projektgruppe von dieser Möglichkeit kein Gebrauch gemacht wird.

## 3.5. jBPM

*Java Business Process Management (jBPM)* [JBoss Community, b] ist ein freies von *jBoss* entwickeltes Business Process Management System (BPMS). BPMS sind Softwaresysteme zur Modellierung und Optimierung von Geschäftsprozessen, die nur Menschen, nur Computer, oder beide involvieren können. Ein BPMS sollte darüberhinaus dazu in der Lage sein, die modellierten Prozesse in eine Prozessdefinitionssprache zu übersetzen, damit diese dann durch einen Computer ausgeführt werden können. *jBPM* ist dabei nicht an eine bestimmte Prozessdefinitionssprache gebunden. *jBPM* unterstützt unter anderem die Prozessdefinitionssprachen *Business Process Execution Language (BPEL)* [XML.org] und *Java Process Definition Language (jPDL)* [JBoss Community, d]. Letztere wird im Rahmen dieser Projektgruppe zur Definition der Prozesse verwendet.

Zum Ausführen der durch die Prozessdefinitionssprache definierten Geschäftsprozesse verwendet *jBPM* die Process Virtual Machine (PVM). Dadurch soll ein gewisser Grad an Zukunftssicherheit erreicht werden, da *jBPM* somit auf recht einfache Art und Weise um zukünftige Prozessdefinitionssprachen erweitert werden kann.

Die einfache Erstellung und Bearbeitung von Geschäftsprozessen unterstützt *jBPM* durch einen graphischen Editor, der als Plugin für die Entwicklungsumgebung *Eclipse* erhältlich ist. Der große Vorteil gegenüber der Erstellung von Geschäftsprozessen direkt im XML-Format liegt vor allem darin, dass damit eine Veranschaulichung der Prozesse möglich ist, die die Zusammenarbeit zwischen Business-Analysten und Entwicklern erheblich erleichtert, da dem Business-Analysten oftmals das technische Verständnis für die zugrundeliegende Prozessdefinitionssprache fehlt.

Einen Überblick über einige weitere Komponenten von *jBPM*, die hier aber nicht weiter erläutert werden sollen, verschafft Abbildung 2.12. Dort ist auch zu erkennen, wie die Process Virtual Machine die Basis für die unterstützten Prozessdefinitionssprachen bildet.

---

<sup>1</sup>Quelle: <http://javabg.eu/2010/02/setting-up-tools-to-build-applications-using-jbpm-part-1/>



Abbildung 3.4.: Komponenten des jBPM-Frameworks<sup>1</sup>

### 3.5.1. Process Virtual Machine

Die Process Virtual Machine kann als das Herzstück von *jBPM* bezeichnet werden. Vergleicht man *jBPM* mit anderen Business Process Management Systemen, die ohne PVM arbeiten, wie z.B. den IBM WebSphere Process Server [IBM], so erkennt man, dass die Nutzung der PVM gewisse Vorteile mit sich bringt. Die im Folgenden erwähnten Vorteile werden dabei zum einen durch eine höhere Flexibilität im Umgang mit Prozessdefinitionssprachen erreicht und zum anderen durch die Unterscheidung der Anwender in Business-Analysten und Entwickler [O'Reilly Media, Inc.].

Viele BPMS legen beim Erstellen von Geschäftsprozessen hohen Wert auf eine leichte Bedienbarkeit durch Business-Analysten, vernachlässigen dabei aber die Notwendigkeit der Einbeziehung von Entwicklern. So sind es vor allem die auf einzelne Einsatzgebiete spezialisierten BPMS, welche durch eine ausgereifte grafische Oberfläche den Fokus auf die Arbeit des Business-Analysten richten. Zu leiden hat darunter dann allerdings der Entwickler, der durch diese strikte Trennung der Arbeit des Business-Analysten von der Arbeit des Entwicklers nur wenig Einblick in den Modellierungsprozess erhält. Durch den eingeschränkten Umfang an Prozessbausteinen, die vom graphischen Editor von *jBPM* zur Verfügung gestellt werden, wird eine engere Zusammenarbeit zwischen Business-Analysten und Entwicklern gefördert. Nur der Entwickler hat durch sein Wissen über den Funktionsumfang der Prozessdefinitionssprache und seine Fähigkeiten in der Programmierung die Möglichkeit, alle durch die *Process Virtual Machine* unterstützten Prozessbausteine im kompletten Umfang zu verwenden. Darüber hinaus kann der Entwickler durch die volle Integration der PVM in die Java Programmierumgebung auf einfache Art und Weise Technologien von Drittanbietern in die Prozesse einbinden, wie beispielsweise Bibliotheken zum Erzeugen von Excel-Dateien oder auch zur Anbindung an soziale Netzwerke.

### 3.5.2. Context

Ein grundlegendes Konzept innerhalb der *PVM* ist der *Context*, welcher es ermöglicht, prozessübergreifend Variablen zur Verfügung zu stellen. Er ist nur während der Ausführung eines Prozesses über die entsprechende Prozessinstanz erreichbar.

Die vom *Context* zur Verfügung gestellten Basisoperationen umfassen das Setzen, Auslesen und Entfernen von Variablen. Dabei werden alle gängigen Java-Variablentypen unterstützt, wie z.B. String, Boolean, Long, Integer, Date, und zusätzlich auch alle Klassen, die über *Hibernate* persistierbar sind. Grundsätzlich ist es auch möglich, beliebige andere Variableninhalte zu verwenden, allerdings resultiert dies dann in einem Fehler beim Speichern des Prozesses in der Datenbank.

Die Variablen sind Teil der Prozessinstanz und werden beim Speichern des Prozesses zusammen mit der Prozessdefinition in die Datenbank geschrieben. Dabei werden die bereits in der Datenbank vorhandenen Variablen mit den Variablen innerhalb der Prozessinstanz abgeglichen, d.h. neue Variablen werden angelegt, vorhandene Variablen werden aktualisiert, gelöschte Variablen aus der Datenbank entfernt.

### 3.5.3. Persistenz

Die Möglichkeit der Persistenz von Prozessdefinitionen und Prozessinstanzen ist ein wichtiger Punkt, wenn es darum geht, Prozesse zu erstellen, die manuelle Eingaben erfordern, um von einem Zustand in den nächsten zu gelangen. Dies trifft zum Beispiel auf sogenannte *wait*-Zustände zu, die eine Eingabe durch einen Mitarbeiter erwarten, um fortzufahren. Da die dabei entstehende Wartezeit beliebig lang sein kann, ist es notwendig, den aktuellen Zustand der Prozessausführung (durch den *Context* beschrieben) in einer Datenbank zu speichern und später bei Bedarf wieder herzustellen.

Das Speichern von Prozessinstanzen, die innerhalb von *jBPM* durch Java-Objekte repräsentiert werden, wird dabei durch das *Hibernate*-Framework realisiert.

### 3.5.4. jPDL

Eine von *jBPM* unterstützte und in dieser Projektgruppe verwendete Prozessdefinitionssprache ist *jPDL* [JBoss Community, d]. Diese Prozessdefinitionssprache spezifiziert ein XML-Schema, das zur Darstellung der Prozesse und der in einem Prozess enthaltenen Tätigkeiten und Zustände verwendet wird, wie auch die Struktur und den Aufbau der Prozessarchivdatei, die die Prozessdefinitionsdatei und weitere zur Prozessdefinition gehörende Dateien (wie beispielsweise Java-Klassen, die das Verhalten der Prozessaktivitäten steuern) beinhaltet.

Ein mit *jPDL* modellierter Prozess besteht aus einer Reihe von Knoten (*Nodes*), Transitionen, die die Knoten miteinander verbinden, und Aktionen (*Actions*), die den Ablauf des Prozesses bestimmen.

Bei der Ausführung eines Prozesses bestimmt ein *Token*, in welchem Knoten sich der Prozess aktuell befindet. Welche Transition im nächsten Schritt gewählt werden soll, wird durch *Signale* gesteuert.

Die meisten Sprachelemente von *jPDL* können direkt innerhalb eines graphischen Editors verwendet werden. Geht es jedoch an speziellere Parametrisierungen dieser Sprachelemente, müssen diese von Hand direkt in der Prozessdatei vorgenommen



werden, die im XML-Format vorliegt.

**Tasks** Knoten vom Typ *Task* repräsentieren Aufgaben, die durch eine Person bearbeitet werden müssen. Dies könnte bspw. die Beschaffung von Ware oder die Ausführung einer Tätigkeit sein. Die Ausführung des Prozesses verharret in diesem Zustand bis die zu bearbeitende Person die ihr zugewiesene Aufgabe erledigt hat. Die Zuweisung einer Aufgabe zu einer bestimmten Person geschieht dabei über Parameter, die bei der Definition des *Tasks* angegeben werden. *Tasks*, die einer bestimmten Person zugewiesen wurden, werden automatisch in der Taskliste der betroffenen Person angezeigt.

Beispiel:

```
<task assignee="alex" name="review">
    <transition name="approved" to="audit"/>
</task>
```

**States** Das Verhalten eines Knotens vom Typ *State* ähnelt dem eines Knotens vom Typ *Tasks*. Auch ein *State* wartet auf das Eintreten eines bestimmten Ereignisses, bevor die Ausführung des Prozesses fortgesetzt werden kann. Allerdings erzeugt ein *State* keinen Eintrag in der Taskliste einer Person. Vielmehr werden *States* verwendet, wenn auf eine Antwort eines externen Systems gewartet werden muss. Das externe System kann dann dem *State* über ein *Signal* mitteilen, dass die Ausführung des Prozesses fortgesetzt werden kann.

Beispiel:

```
<state name="audit">
    <transition name="to_end" to="end"/>
</state>
```

**Forks and Joins** Mit *Forks* und *Joins* kann man in jPDL die Ausführung von parallelen Tasks modellieren. Dabei wird ein Knoten vom Typ *Fork* dazu verwendet, den Prozessfluß an dieser Stelle aufzuteilen, und mit einem Knoten vom Typ *Join* kann der parallele Prozessfluß wieder zusammengeführt werden. Dabei wartet die Ausführung der Prozessinstanz im *Join*-Knoten, bis alle parallelen Zweige abgearbeitet wurden.

**Decision** Mit Knoten von Typ *Decision* können Entscheidungen modelliert werden, die der Prozess allein anhand von ihm zugänglichen Informationen treffen kann. Dies können beispielsweise das Abfragen und Vergleichen von Variablenwerten aus dem *Context* sein.

Setzt eine Entscheidung allerdings das Eingreifen einer Person voraus, so sollte dies stattdessen mit einem *Task* modelliert werden, der mehrere ausgehende *Transitions* besitzt, die den möglichen Ergebnissen der Entscheidung entsprechen.

**Custom** Knoten vom Typ *Custom* sind spezielle Knoten, die einzig dazu dienen, eine Methode aus einer Java-Klasse aufzurufen. Dafür muss die Java-Klasse ein spezielles Interface implementieren.

**Transitions** Mit *Transitionen* werden die verschiedenen *Nodes* eines Prozesses miteinander verbunden und somit Pfade der Ausführung dieses Prozesses spezifiziert. *Transitionen* können beschriftet werden, falls mehrere mögliche Pfade aus einem *Node* heraus existieren, wie es bei Knoten vom Typ *Decision* der Fall ist.

### 3.5.5. Graphical Process Designer

*jBPM* stellt mit dem Graphical Process Designer (GPD) einen graphischen Editor für Prozesse als Plugin für die Entwicklungsumgebung *Eclipse* zur Verfügung. Dieses Plugin stattet *Eclipse* mit der Fähigkeit aus, *jBPM*-Projekte zu erstellen. Dabei werden beim Anlegen eines neuen Projekts automatisch alle nötigen Verzeichnisse erzeugt und die benötigten Klassenpfade korrekt gesetzt. Zum Erstellen einer neuen Prozessdefinition kann anschliessend ein Wizard aufgerufen werden, der initial eine XML-Datei anlegt, in der die Prozessdefinition gespeichert wird.

Die Modellierung der grundlegenden Struktur des Prozesses wird in der Regel in der graphischen Ansicht vorgenommen. Per Drag&Drop können die verschiedenen Bausteine eines Prozesses in den Editor gezogen und miteinander verbunden werden. Zeitgleich werden alle Änderungen automatisch in die Prozessdefinitionsdatei übernommen. Durch einen Reiter kann zu jeder Zeit zwischen beiden Ansichten gewechselt werden (siehe Abbildung 3.5).

### 3.5.6. Beispielprozess

Ein beispielhafter Prozess, der im Graphical Process Designer modelliert wurde, wird in Abbildung 3.6 gezeigt. Der Prozess stellt den Ablauf einer Kundenanfrage dar, zu deren Beginn zunächst entschieden wird, ob der anfragende Kunde bereits bekannt ist. Diese Aufgabe übernimmt ein Knoten vom Typ *Decision*. Ist der Kunde noch nicht bekannt, so wird ihm von einem Sachbearbeiter ein Kundenbetreuer zugewiesen. Da dafür ein Eingreifen einer Person notwendig ist, wird dieser Knoten anhand eines *Tasks* modelliert. Wurde dem Kunden ein Betreuer zugewiesen, so wird der Kunde daraufhin mit Hilfe eines Java-Knotens in der Datenbank gespeichert. Die nächsten Schritte umfassen das Bearbeiten der Kundenanfrage und das

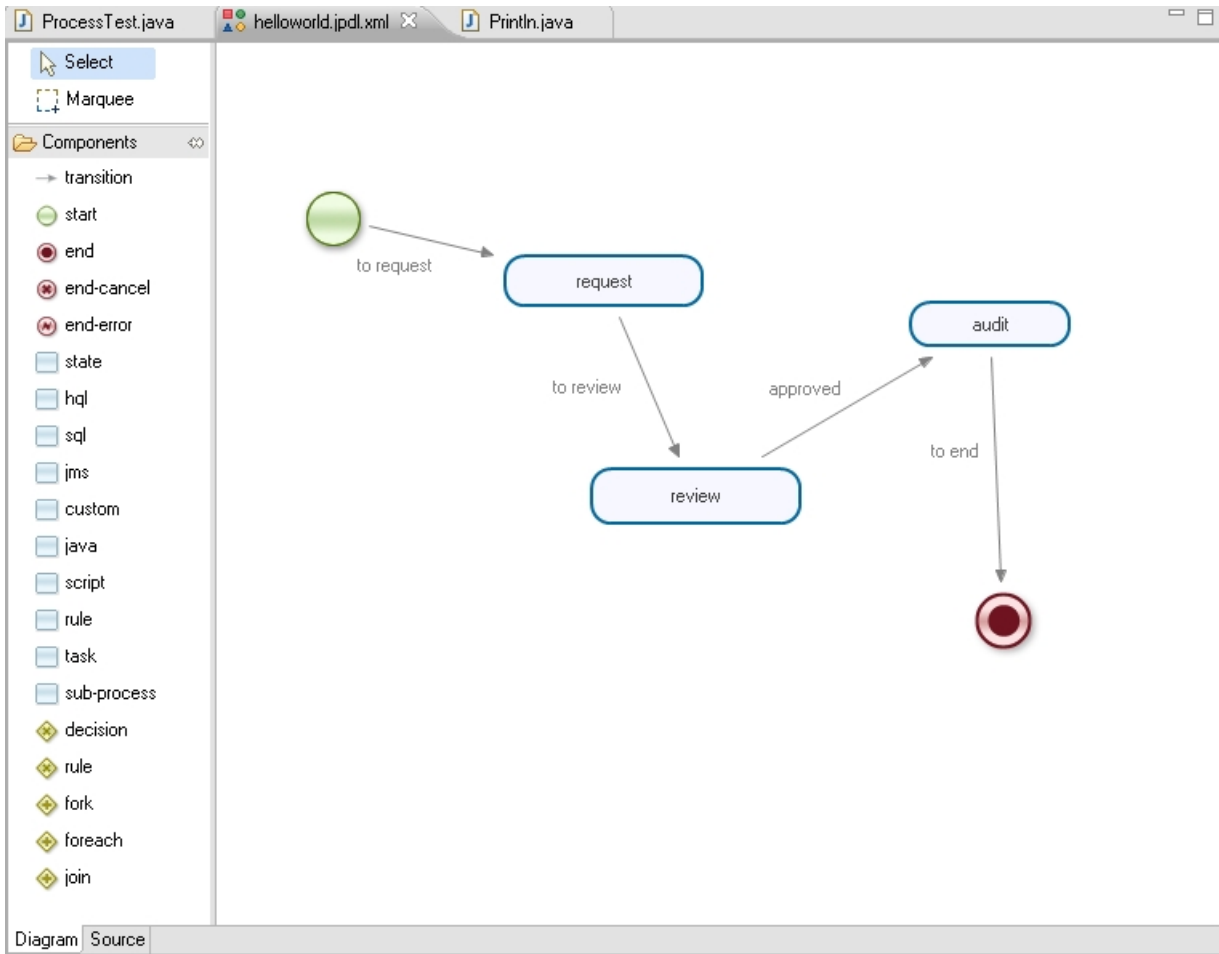


Abbildung 3.5.: Graphical Process Designer

Benachrichtigen des Kunden. Auch für diese beiden Schritte werden Knoten vom Typ Task verwendet, da es sich auch hierbei wieder um Tätigkeiten handelt, die das Eingreifen einer Person erwarten. Nachdem der Kunde benachrichtigt wurde, wartet der Prozess in einem Knoten vom Typ State auf eine Antwort des Kunden.

Wenn man sich nun die XML-Repräsentation des Prozesses anschaut, erkennt man, dass erst dort ersichtlich wird, um welchen Typ es sich bei den Knoten aus Abbildung 3.6 eigentlich handelt.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<process name="customercare" xmlns="http://jbpm.org/4.4/jpdl">
  <start name="Supportanfrage">
    <transition to="Kunde bekannt?"/>
  </start>
  <decision name="Kunde bekannt?">
```

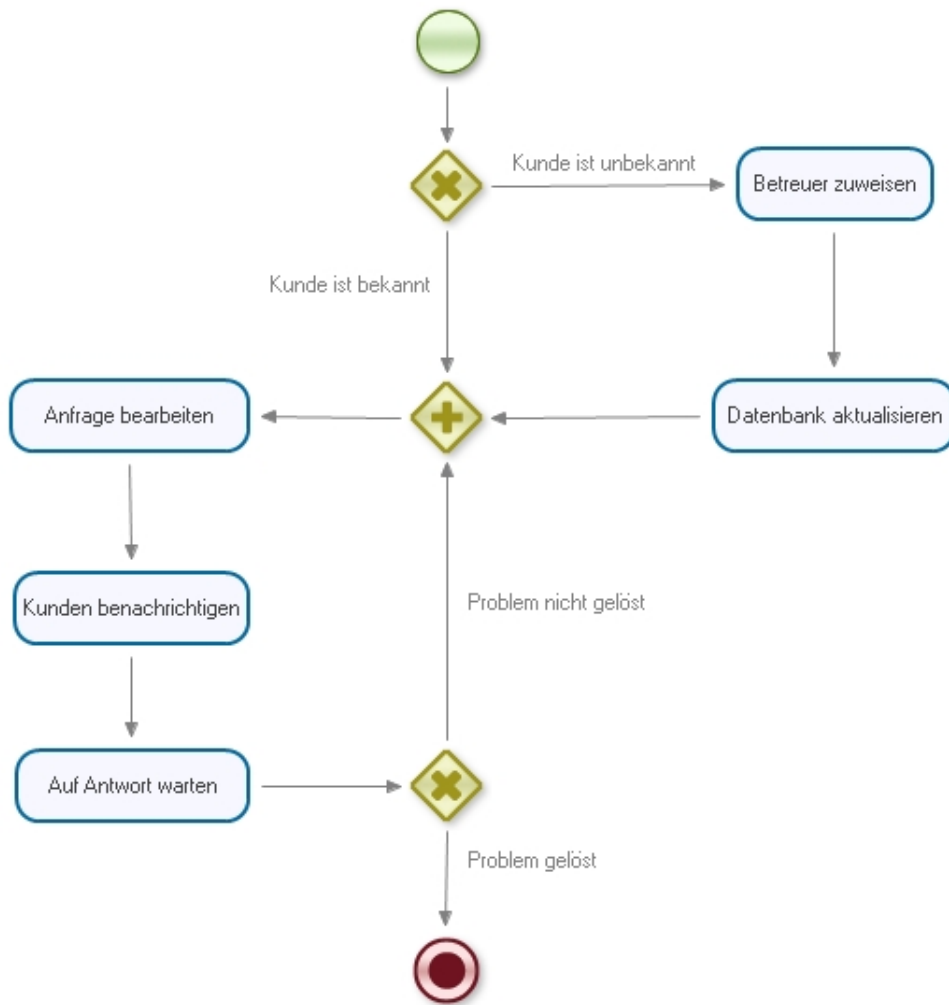


Abbildung 3.6.: Beispielprozess im Graphical Process Designer

```

    <transition name="Kunde ist unbekannt" to="Betreuer zuweisen"/>
    <transition name="Kunde ist bekannt" to="Kunde ist bekannt"/>
</decision>
<task name="Betreuer zuweisen">
    <transition to="Datenbank aktualisieren"/>
</task>
<java class="de.mycompany.customercare.Customers" method="addCustomer"
    name="Datenbank aktualisieren">
    <transition to="Kunde ist bekannt"/>
</java>
<join name="Kunde ist bekannt">
    <transition to="Anfrage bearbeiten"/>
</join>

```

```

<task name="Anfrage bearbeiten">
  <transition to="Kunden benachrichtigen"/>
</task>
<task name="Kunden benachrichtigen">
  <transition to="Auf Antwort warten"/>
</task>
<state name="Auf Antwort warten">
  <transition to="Kunde zufrieden?"/>
</state>
<decision name="Kunde zufrieden?">
  <transition name="Problem nicht gelöst" to="Kunde ist bekannt"/>
  <transition name="Problem gelöst" to="Ende"/>
</decision>
<end name="Ende"/>
</process>

```

### 3.6. Scrum

Menschen, die Software entwickeln, oder den Entwicklungsprozess begleiten, sind im Allgemeinen in einem Team organisiert und müssen entsprechend koordiniert werden. Auch die zunehmende Spezialisierung, eine zwangsläufige Reaktion auf die Komplexität heutiger Softwaresysteme, verschärft den Bedarf an weiterer Organisation. Dieser Prozess hat bekanntlich einen eigenen Forschungszeitweig begründet, die Softwaretechnik.

Im Bereich der Herstellung von Software haben sich dabei gewisse Standards etabliert, die Vorgehensmodelle. Die ältesten davon, die sich im Laufe der letzten 40 Jahre durchgesetzt haben, nennt man klassische Modelle. Als Prototyp für solche klassischen Modelle lässt sich hier zweifellos das Wasserfallmodell nennen [Royce, 1987]. Aus Mangel an Flexibilität und aufgrund einer gewissen Schwerfälligkeit dieser Modelle haben in jüngster Zeit sogenannte agile Methoden Einzug in die Softwareentwicklung gehalten. Vor allem Scrum [Pichler, 2008], ein typischer Vertreter dieser Methoden, hat in den letzten Jahren einen Siegeszug gefeiert und erfreut sich wachsender Beliebtheit.

Agile Softwareentwicklung versteht sich als Gegenbewegung zu den schwergewichtigen klassischen Vorgehensmodellen und wurde maßgeblich inspiriert von Entwicklungen in der japanischen Automobilbranche (allen voran Toyota) mit Beginn der zweiten Hälfte des 20. Jahrhunderts. Diese erschaffte neue Konzepte der Produktionsentwicklung, die heute unter dem Begriff des „Lean Production“ (schlanke Produktion) bekannt sind, und nicht mehr nur in der Fahrzeugbranche Verwendung finden. Im Zeichen von Wertschöpfung und der Vermeidung von Verschwendung konzentrierte man sich auf die wichtigen Ziele, wie Kompetenzen zusammen zu führen und Abläufe zu harmonisieren. Der Entwicklungsprozess sollte im Ganzen schlanker und flexibler werden [Gloger, 2008].

Der Einfluss auf andere Branchen machte auch vor der Softwareentwicklung keinen Halt und so trafen sich im Jahr 2001 interessierte Entwickler auf einer Konferenz zu diesem Thema, nachdem wenige Jahre zuvor schon erste Erfahrungen mit agilen Prozessen gesammelt wurden. Dort legte man den Grundstein für eine gemeinsame Sprache, einigte sich auch auf den Begriff „agil“ und erklärte das agile Manifest [Agile Alliance, 2001].

Das Agile Manifest versucht die Werte und Ideen agiler Softwareentwicklung auf den Punkt zu bringen und zu verbreiten. Die wichtigsten vier Kernforderungen sind:

1. Individuen und Interaktionen über Prozesse oder Werkzeuge
2. Lauffähige Software über umfangreiche Dokumentation
3. Zusammenarbeit mit Kunden über Vertragsverhandlungen
4. Auf Veränderungen reagieren über eine strikte Planverfolgung

Der Entwicklungsprozess von Softwareprodukten ist meist so komplex, dass eine vorläufige Betrachtung aller Eventualitäten durch eine umfassende und endgültige Planung eigentlich nicht möglich ist. Deshalb versucht Scrum, ähnlich wie andere agile Prozesse, eine ganzheitliche Planung zu vermeiden. Anstatt zukünftige Aufgaben, und somit alle Arbeitsschritte, in entsprechenden Phasen vorausszusehen, wird auf Anforderungen und Ereignisse immer wieder neu eingegangen [Pichler, 2008]. Dabei entsteht die Software durch einen iterativen Prozess, der es ermöglicht den Projektfortschritt genau zu beobachten und zu kontrollieren. Hohe Qualitätsansprüche an die Software, Effizienz, Kundenzufriedenheit aber genauso eine Unternehmenskultur, die die Mitarbeiter zufrieden stellt, sind die Intentionen von Scrum. Hier liegen die Maßstäbe für eine erfolgreiche Durchführung von Scrum [Pichler, 2008].

### **3.6.1. Rollen in Scrum**

Um eine agile und effiziente Projektdurchführung zu erreichen, wird bei Scrum eine klare und einfache Rollendefinition aller Beteiligten gefordert. In einer flachen Hierarchie wird größtmöglicher Wert auf Selbstorganisation und Eigenverantwortlichkeit gelegt. Dabei unterscheidet man grundsätzlich drei verschiedene Rollen: den Project Owner, das Team und den Scrum Master. Der Project Owner ist der Hauptverantwortliche im System. Er überwacht den Entwicklungsprozess durch einen betriebswirtschaftlichen und ingenieuren Blick auf das Ergebnis. Er erstellt den Product Backlog, eine Art To-Do-Liste, und priorisiert dessen Einträge, die Anforderungen an das Produkt.

Das Team ist für die eigentlich zu verrichtende Arbeit zuständig. Alle Aufgaben, die während der Produkterstellung anfallen, werden durch die Mitglieder implementiert bzw. kreiert. Das Team arbeitet dabei in höchstem Grade eigenverantwortlich und wählt die aktuellen Aufgabenpakete selbst aus, die es im Laufe eines Sprints,

ein kurzer aber vollständiger Entwicklungszyklus, umsetzen will. Es schätzt die Aufwände und ist als Gesamtheit für die Erreichung der selbstauferlegten Pflichten zuständig.

Der Scrum Master ist der gute Hirte des Teams. Er ist für die korrekte Durchführung des Scrumprozesses zuständig und erinnert alle Beteiligten an ihre Rollendefinitionen. Er unterstützt das Team in ihrer Organisation und kümmert sich um einen reibungsfreien Ablauf. Der Scrum Master dient dem Team zum Schutz vor äußeren Einflüssen [Pichler, 2008]. Bereits von Beginn an wird eine Integration mit allen Projektinteressierten angestrebt. Das können Vertrieb, Management, Marketing und Kunden sein. Vor allem die Kunden, die später von der Softwareentwicklung profitieren sollen, haben dabei ein entscheidendes Mitspracherecht. Sie können den Projektfortschritt lenken und beeinflussen. Zeitnahe Feedbacks und Workshops helfen die Wünsche und Pflichten zu erkennen und umzusetzen.

### 3.6.2. Artefakte in Scrum

Scrum stehen eine Reihe von Werkzeugen oder Methoden zur Verfügung, die helfen Scrum richtig zu organisieren und die Übersicht über den Projektfortschritt zu behalten. Diese Artefakte sind generell öffentlich und von jedermann einsehbar, nicht nur den Beteiligten. Das hilft die Transparenz zu wahren. Dazu gehören:

- Product Backlog
- Sprint Backlog
- Releaseplan
- Benutzergeschichten (User Stories)
- Burndown-Berichte

Vor allem die User Stories kamen beim Projektmanagement der PG zum Einsatz. User Stories sind dabei keine Erfindung von Scrum, haben sich hier, wie auch in anderen agilen Methoden, allerdings durchgesetzt [Cohn, 2004]. Jede User Story enthält in der Regel die folgenden vier Informationen:

1. Wer ist der Handelnde der Story? (Rolle)
2. Was möchte der Handelnde tun? (Aktivität)
3. Was möchte der Handelnde erreichen? (Ziel)
4. Testkriterien für die beschriebene Story

User Stories sind bewusst einfach gehalten, ohne Anspruch auf vollständige Dokumentation zu erheben. Gerade der informale Charakter unterstützt einen regen Informationsaustausch zwischen Product Owner, dem Ersteller der Geschichte in Korrespondenz mit dem Kunden und dem Team als Adressat. Außerdem versucht man damit konkrete Designentscheidungen nicht zu beeinflussen. Es wird vermieden Merkmale der Benutzerschnittstellen in die Spezifikation aufzunehmen, wobei hier das Wort Spezifikation eigentlich schon über das Ziel hinaus schießt. User Stories sollen ja gerade Anforderungen repräsentieren, nicht dokumentieren.

Die anderen Artefakte sind nicht minder wichtig. Im Gegenteil, sie haben sogar eine zentralere Funktion als die optionalen Userstories. Dennoch werden sie hier nur kurz erläutert, da sie für unsere Planungen keine Rolle spielen. Für eine ausführliche Betrachtung dieser Werkzeuge verweisen wir auf [Gloger, 2008], [Pichler, 2008] oder [Schwaber, 2004].

Die beiden Backlogs (Product Backlog und Sprint Backlog) dienen der Spezifikation der Software, denn auch hier müssen - auch wenn es sich um eine agile Vorgehensweise handelt - Dinge fixiert werden. Während der Product Backlog die gesamte Software im Auge behält und eine komplette, meist umgangssprachliche Beschreibung des Endprodukts umfasst, beschäftigt sich der Sprint Backlog mit den technischen Aspekten der verschiedenen Iterationen. Deshalb gibt es auch mehrere davon - für jeden Sprint einen.

Releaseplan und Burndown-Bericht dienen der Einhaltung von Terminen und geben Übersicht über den Projektfortschritt. Der Releaseplan ist eine Gliederung des Product Backlogs in eine Reihe von möglichen Sprints. Er wird noch vor dem ersten Sprint vom Product Owner erstellt und dann stetig weiterentwickelt. Hier werden vor allem Zeit und Kosten in Einklang gebracht. Burndown-Berichte sind Grafiken und dienen der Sichtbarmachung von Fortschritten im Projekt. Traditionelle Projektstatusberichte gibt es in Scrum nicht, deshalb dienen diese Elemente als Beurteilung des Entwicklungsflusses.

### **3.6.3. Ablauf**

Jedes Projekt, welches mit Scrum organisiert wird, durchläuft spezifische Phasen (bzw. deren zugrunde liegenden Meetings), die einen wesentlichen Anteil an der Organisation haben. Dazu zählen:

- Sprint
- Sprint-Planungssitzung
- Daily Scrum
- Sprint-Review
- Sprint-Retrospektive



Die wichtigste Phase ist zweifelsfrei der Sprint. Als Sprint bezeichnet man einen einzelnen Entwicklungszyklus. Ziel dieses Entwicklungsschritts ist ein funktionierendes, auslieferbares Stück Software, das sowohl getestet als auch dokumentiert ist. Dieses nennt man Produktinkrement. Das ganze Projekt durchläuft dabei mehrere solcher Sprints, bis am Ende das fertige Softwareprodukt steht (s. Abbildung 3.7).

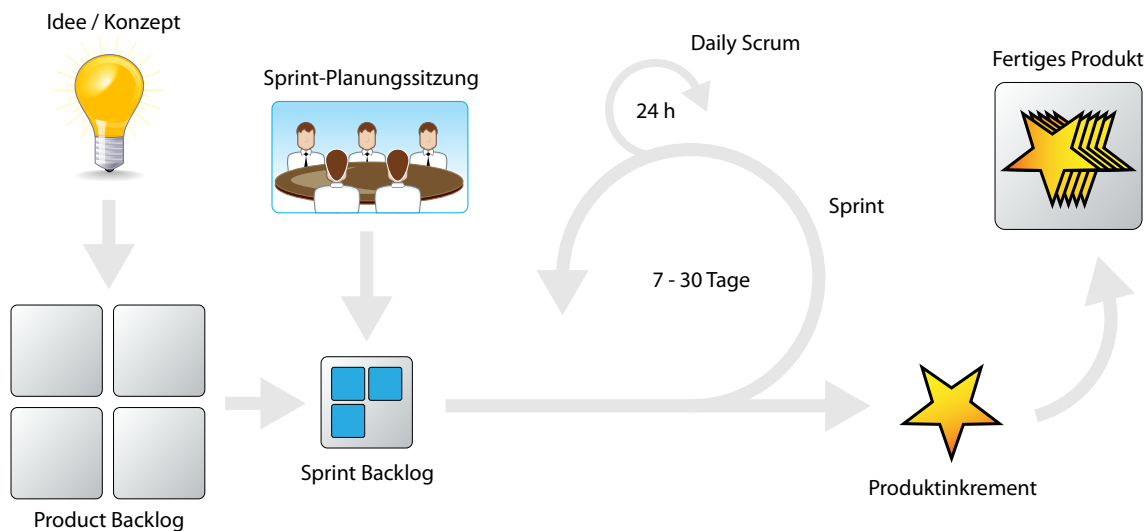


Abbildung 3.7.: Der typische Ablauf von Scrum

Ein Sprint dauert in der Regel 7 - 30 Tage, je nach Projektanforderungen. Zweiwöchige Sprints haben sich in der Praxis gut bewährt [Pichler, 2008]. Die Planung zu Beginn eines jeden Sprints wird in der Sprint-Planungssitzung durch Project Owner, Team und Scrum Master abgeschlossen. Das Team legt hier genau fest, welche Anforderungen innerhalb des Sprints in ein Inkrement umgewandelt werden. Dazu identifiziert es die entsprechenden Aktivitäten und schätzt deren Aufwände. Diese Informationen fließen in den Sprint Backlog und sind für alle einsehbar (s. Abbildung 3.7). Genauso wie sich das Team verpflichtet die Anforderungen innerhalb des Sprints zu erfüllen, werden zu seinem Schutz alle Inhalte für die Dauer des Sprints fixiert. Nur so kann sich das Team erfolgreich selbst organisieren.

Für Sprints gilt eine strikte Termineinhaltung. Start- und Endtermine werden nie verschoben oder verändert. Sollte das Team seine Ziele nicht einhalten können, werden gewisse Anforderungen in Absprache mit dem Product Owner in den nächsten Sprint verschoben. Hier ist es von Vorteil, dass alle Einträge im Backlog priorisiert sind und Wichtiges von Unwichtigem unterscheidbar ist. Wird widererwartend ein schnellerer Fortschritt erreicht und ist das Team bereits vor Ende des Sprints mit allen Aufgaben fertig, bittet das Team den Product Owner um weitere Aufgaben aus dem Product Backlog. Hier gilt, genau wie für den normalen Verlauf: Ein Team muss in der Lage sein die Anforderungen erfolgreich und termingerecht umsetzen zu können. Sollten neue Anforderungen nicht mehr in den aktuellen Sprint hineingezogen werden können, so kann die verbliebene Zeit auch genutzt werden, um die

Qualität des Codes zu erhöhen, beispielsweise durch Refaktorisierung oder erweiterte Modultests.

Der interessierte Leser findet eine ausführliche Betrachtung aller Phasen in [Gloger, 2008], [Pichler, 2008] oder [Schwaber, 2004].

## 4. Projektmanagement

Die Arbeit in der Projektgruppe erfordert selbstverständlich eine Organisation aller Beteiligten, sowohl was zeitliche Aspekte betrifft, als auch den Ablauf und die Vorgehensweise. Während ein grober Rahmen bereits von den Regularien der Fakultät Informatik (bzgl. der Projektgruppe) vorgegeben ist, ist die interne Verwaltung eine Angelegenheit der studentischen Mitglieder. Vor allem der Softwareentwicklungsprozess bedarf einer konkreten Organisation - was man gemeinhin in der Softwareentwicklung als Vorgehensmodell bezeichnet.

### 4.1. Termine

Bereits zu Beginn der Arbeitsaufnahme wurde von den Teilnehmern der Projektgruppe (PG) ein Termin festgelegt, an dem sie sich in einem wöchentlichen Rhythmus treffen. Dieser Pflichttermin mit den Betreuern dient dem Berichterstaten und der Fortschrittsanzeige der Projektgruppe. Zugleich werden hier oft wichtige Entscheidungen getroffen. Zwei weitere Termine in der Woche dienen dem gemeinschaftlichen Arbeiten. Hier finden oft Gruppendiskussionen statt und es wird an den Implementierungen der verschiedenen Projekte gearbeitet. Ansonsten wird von zuhause aus gearbeitet und per Email oder Instant Messenger kommuniziert.

Im Laufe des ersten Semesters haben wir festgestellt, dass eine *lockere Berichterstattung* bei unseren wöchentlichen Treffen mit den Betreuern nicht optimal war. Aus diesem Grund haben wir gleich zu Beginn des zweiten Semesters beschlossen unsere Meetings dahingehend zu ändern, dass jeder Teilnehmer zu Beginn eines jeden Meetings berichtet woran er oder sie seit dem letzten Treffen gearbeitet hat und welche Probleme dabei eventuell aufgetreten sind. Dies war vorteilhaft, da jeder Teilnehmer nicht ausschließlich die ihm zugewiesenen Aufgaben bearbeitet hat, sondern darüber hinaus eventuell auch schon weitere Aufgaben oder aufgetretene Probleme bearbeitete. So konnten eventuell noch nicht gelöste Probleme in der Gruppe besprochen und teilweise auch bereits gelöst werden.

Zusätzlich mussten wir leider feststellen, dass eine vollkommen selbstständige Zuweisung der identifizierten Aufgaben selten funktionierte. Deshalb wurde beschlossen, dass jedes Meeting mit einer Aufgabenverteilung enden sollte, wo jedem Teilnehmer eine entsprechende Aufgabe zugewiesen wurde, welche er oder sie bis zum nächsten Treffen bearbeiten sollte. Um jedem Teilnehmer noch ein gewisses Maß an Eigenverantwortung zu überlassen wurden hier allerdings lediglich grobe Aufgabenbeschreibungen verfasst, so dass alle zu bearbeitenden Aspekte zugewiesen sind. Wir haben hier gezielt auf grobe Aufgabenbeschreibungen gesetzt, um die Aufgaben-

verteilung möglichst effizient durchführen zu können. Die Aufgliederung in mehrere tatsächlich durchführbare, kleinere Aufgaben bzw. Einzelschritte blieb dem jeweiligen Teilnehmer überlassen.

## 4.2. Vorgehen à la Scrum

Den größten Teil der Organisation macht allerdings das Vorgehensmodell aus. Wie in den Grundlagen bereits beschrieben wurde (siehe Abschnitt 3.6 zum Thema Scrum), ist es unverzichtbar, den reibungslosen Ablauf einer Softwareentwicklungsarbeit entsprechend zu organisieren. Dabei sollte ein vernünftiges Vorgehensmodell angewandt, sowie geeignete Softwaretools zur Verfügung gestellt werden. Als Ausgangspunkt dient der PG hier Scrum (siehe Abschnitt 3.6), da es eine sinnvolle und vor allem agile Entwicklung der Software ermöglicht. Die Agilität des Modells fördert den Zusammenhalt der Gruppe und behält den Fokus auf wesentliche Teile der Entwicklerarbeit. So entspricht die Gruppengröße ungefähr der in Scrum definierten. Außerdem sollte die Projektgruppe - genau wie das Team in Scrum - möglichst selbstorganisiert vorgehen. Die Betreuer der PG verzichteten bewusst auf eine administrative Steuerung ihrerseits.

Eine vollständige Umsetzung von Scrum ist im Rahmen einer PG allerdings nicht möglich. Das liegt zum einen am Faktor Zeit. Ein Projekt, welches mit Scrum entwickelt wird, setzt eine Vollzeitbeschäftigung aller Beteiligten voraus. Bei der zur Verfügung stehenden Anzahl von Semesterwochenstunden ist das aber kaum möglich. So sind zum Beispiel die täglichen, unverzichtbaren Treffen (Daily Scrum) in der Form gar nicht möglich. Außerdem fehlt der PG die Rolle des Scrum Masters, der das Team im Umgang mit Scrum unterstützen soll. Das liegt vor allem daran, dass keiner der Teilnehmer eine praktische Erfahrung mit Scrum hat und so geeignete Methoden und Mittel zur Verfügung stellen kann. Die Rolle des Project Owner ist allerdings weniger vernachlässigbar. Gegen Ende des ersten Semesters hat die PG deshalb einen solchen Projektleiter basisdemokratisch bestimmt. Zu den Aufgaben des Projektleiters gehören die Projektorganisation und -koordination. Die Kommunikation innerhalb des Teams wird so sichergestellt und vorangetrieben. Außerdem hat er maßgeblichen Einfluss auf die Aufgabenverteilung (Wer macht was?) und die Prioritätsverteilung (Was wird zuerst gemacht?). Explizite Deadlines werden durch seine Gesamtsicht auf das Projekt nicht aus den Augen verloren. So ist eine gleichmäßige Aufgabenverteilung und strukturiertere Abarbeitung der einzelnen Aufgaben gewährleistet.

Was von Scrum übrig bleibt ist vor allem die selbstorganisierende Struktur und das agile Vorgehen bei der Entwicklung der Software. Unser Bestreben war es immer eine möglichst ausführbare und getestete Version des Produktes vorliegen zu haben. Dabei wurden die wöchentlichen Pflichttreffen der PG sozusagen als ein einzelner Sprintzyklus betrachtet und die Anwendung(en) bei diesen Treffen, wenn es nötig erschien, demonstriert. Außerdem hat die PG die Idee der Userstories in ihren Workflow integriert. Diese stellen eine Möglichkeit dar, die Anforderungen an die Software

in umgangssprachliche Formulierungen zu pressen und nicht sofort zu technisch an die Sache heranzugehen. Ein dadurch entstandener allgemeiner Diskurs bzw. Disput ist so möglich und fruchtbar. Um den Stand der Software zu dokumentieren und Aufgaben schnell verteilen zu können, setzte die PG auch eine Art Backlog (Konglomerat aus Sprint Backlog und Product Backlog) ein, welches vergleichbar mit einer einfachen Todo-Liste ist.

### 4.3. Konkretes Vorgehen

Alle Aufgaben des Teams wurden auf verschiedene Teilgruppen aufgeteilt. So gab es - entsprechend der Architektur (siehe Kapitel 5) - eine Teilgruppe, die sich mit der Daten- und Serviceschicht beschäftigte, eine Gruppe für die Prozessschicht und eine Fraktion für das Userinterface respektive die Webschicht. Eine Einheit war außerdem für die Anforderungsanalyse zuständig. Die Mitglieder der Gruppen sind dabei nicht statisch. Während der Arbeit war es durchaus üblich, sich anderen Gruppen anzuschließen, je nachdem wie stark der Arbeitsanfall gerade in der entsprechenden Gruppe war und Unterstützung gebraucht wurde. Die von der PG abgespeckte Version von Scrum begann immer mit einer Anforderungsanalyse. Hier ist nochmal zu erwähnen, dass diese Projektgruppe aus mehreren kleinen Teilprojekten bestand und dieser Prozess mehrmals durchlaufen wurde.

In dieser Analyse wurde der Kunde (siehe Kapitel 2) aufgesucht und seine Vorstellungen und Wünsche für die Software festgehalten. Diese Informationen wurden anschließend in der entsprechenden Teilgruppe diskutiert und in Userstories übersetzt, damit auch die anderen Teammitglieder einen Überblick über die Aufgaben erhalten. Die Userstories sind dabei typischerweise nach Prioritäten sortiert und können auch während der Umsetzung noch angepasst werden. Dazu ist es auch gegebenenfalls nötig, weitere Rücksprache mit dem Kunden zu halten und Inhalte - wenn die Zeit gekommen ist - genauer zu spezifizieren.

Hier zeigt sich wieder das agile Vorgehen unseres Ansatzes. Nachdem die Userstories für jedermann zugänglich gemacht sind, werden entsprechende Aufgaben bzw. Tasks in den Teilgruppen identifiziert und führen so zu einer Todo-Liste. Zur Administration der Todo-Liste und der Userstories wurde anfänglich Origo [ETH Zürich] eingesetzt. Später hat sich die PG zusätzlich für den Bugtracker PivotalTracker [Pivotal Labs, Inc.] entschieden. Gegen Ende des zweiten Projekts wurde jedoch endgültig auf das Projektmanagement-Tool Redmine [Lang] (s. Abschnitt 4.4 umgestiegen, welches alle unsere Bedürfnisse in einem Tool vereinigt.

### 4.4. Redmine

Redmine ist ein webbasiertes Projektmanagement-Tool, in welchem es möglich ist einzelne Tasks (Aufgaben) anzulegen und diese einzelnen Personen zuzuweisen. Jeder Task besitzt einen Bearbeitungsstatus und eine Anzeige des Fortschritts, sodass

der Verlauf einer Aufgabe leicht verfolgt werden kann. Zusätzlich kann jeder Task kommentiert werden, was eine Diskussion und Dokumentation eventueller Schwierigkeiten ermöglichte. Um eine ungefähre Vorstellung bezüglich des Projektstatus zu erhalten, wurde festgelegt, dass jeder Teilnehmer, der beginnt einen Task zu bearbeiten den Status auf *In Bearbeitung* und den Fortschritt auf *10%* setzt. Sobald der Task, aus Sicht des Teilnehmers, erledigt war, wurde der Status auf *QC (Qualitätscheck)* und der Fortschritt auf *90%* gesetzt. Der Fortschritt zwischen *10%* und *90%* wurde nach eigener Einschätzung vom Bearbeiter selbst gewählt. Sobald ein Task auf *QC* gesetzt wurde, konnten andere Teilnehmer die Arbeit überprüfen und den Task auf *100%* und *Erledigt* setzen. Wurde noch ein Fehler gefunden, so wurde dies in einem Kommentar vermerkt und der Status zurück auf *In Bearbeitung* und der Fortschritt auf *80%* gesetzt. Diese Konventionen halfen uns dabei einen einigermaßen realistischen Überblick über den Fortschritt des Projektes zu behalten.

Ein weiteres Feature von Redmine erlaubt es zu gegebenen Tasks weitere untergeordnete Tasks zu erstellen, welches wir dazu genutzt haben die zu Beginn definierten Userstories in kleinere Tasks aufzuteilen. Dabei sind wir so vorgegangen, dass die anfangs definierten Userstories als Tasks angelegt wurden. Wurde dann beschlossen, dass eine bestimmte Userstory umgesetzt werden sollte, so wurden dem Task, welcher der Userstory entsprach, entsprechende untergeordnete Tasks hinzugefügt, welche dann implementiert wurden.

Zusätzlich bietet Redmine die Möglichkeit einzelnen Tasks bestimmte Versionen zuzuordnen. Dies haben wir dazu genutzt, eine Version für jede Woche zu erstellen. Über eine sogenannte Roadmap, welche alle noch offenen Versionen auflistet, konnten wir leicht einen aktuellen Projektstatus ermitteln.

## 4.5. Konventionen

Im Verlauf des ersten Semesters hat sich gezeigt, dass es unabdinglich ist einen gewissen Rahmen festzulegen, in welchem jeder arbeiten sollte. Darunter fällt zum einen das Task-Management, welches im Abschnitt 4.4 bereits erläutert wurde, aber ebenso Code- und VCS-Konventionen.

### 4.5.1. Code-Konventionen

Während der ersten beiden Projekte ist uns aufgefallen, dass es ziemlich schwierig sein kann Code zu lesen, der von verschiedenen Personen geschrieben wurde. Aus diesem Grund haben wir zu Beginn des zweiten Semesters einige Code-Konventionen festgelegt, die bestimmen, wie Kommentare, Variablen- und Methodenbezeichner zu schreiben sind. Ebenso wurden Regeln zur Einrückung von Code festgelegt, was den Code des dritten Projektes deutlich lesbarer gestaltete. Auf die genauen Code-Konventionen soll hier jedoch nicht weiter eingegangen werden, da dies stark von den Vorlieben und der Arbeitsweise der jeweiligen Teilnehmer abhängig ist.

## 4.5.2. VCS-Konventionen

Zusätzlich zu den bereits genannten Konventionen haben wir im ersten Semester diverse Probleme im Umgang mit der Versionsverwaltung festgestellt. Dies beginnt bereits bei der Einrichtung des Repositories. Bei der Verwendung von Subversion (SVN) ist es üblich zunächst die Verzeichnisse *branches*, *tags* und *trunk* anzulegen, wobei *trunk* den aktuellen Entwicklungszweig beinhaltet. Da wir jeweils mit drei Teilprojekten für die Service-, Prozess- und Webschicht gearbeitet haben, sah unser Setup für die ersten beiden Projekte so aus, dass wir auf oberster Ebene zunächst Unterordner für jedes Teilprojekt und dort dann die Ordner für *branches*, *tags* und *trunk* angelegt haben. Dies hatte eine im Nachhinein umständliche Behandlung der Projekte zur Folge, da Projekte immer einzeln ausgecheckt, manche Änderungen aber projektübergreifend durchgeführt werden mussten. Dadurch mussten eigentlich zusammenhängende Commits in mehrere Commits (je Teilprojekt) aufgeteilt werden. Die Einrichtung für das dritte Projekt im zweiten Semester wurde dahingehend geändert, dass nun auf oberster Ebene die Unterscheidung nach *branches*, *tags* und *trunk* vorgenommen wurde, und darin dann jeweils alle Teilprojekte verwaltet wurden. Dies ermöglichte uns nun zusammenhängende Änderungen in einem Commit projektübergreifend durchzuführen.

Darüber hinaus wurde festgelegt, dass zu jedem Commit vermerkt wird, zu welchem Task dieser gehört, die Task-ID also direkt mit in den Commit-Text einzutragen. Auf diesem Wege konnte schnell nachvollzogen werden zu welcher Aufgabe ein bestimmter Commit gehört und um welches Problem es eigentlich geht. Dieses Vorgehen half uns bei auftretenden Problemen die problematischen Änderungen schneller zu finden und strukturierter beheben zu können.





# 5. Architektur

Die Architektur, auf Basis derer die Software entwickelt wurde, wird im Folgenden anhand eines Schichtenmodells genauer erläutert. Auch in der Auswahl der Tools für den Aufbau der Entwicklungsumgebung ist dieses Schichtenmodell erkennbar. Auf die einzelnen Tools und die Begründung zu ihrer Auswahl wird in diesem Zuge ebenfalls eingegangen.

## 5.1. Schichtenmodell

Die Projektgruppe verwendet die in Java EE (siehe Kapitel 3.1) vorgegebenen Schichten mit leichten Änderungen. Das verwendete Schichtenmodell ist in Abbildung 5.1 abgebildet und wird im folgenden erläutert. Als erstes gibt es eine Präsentationsschicht (1) welche mittels Tapestry (siehe Kapitel 3.4) dem Benutzer eine Web-Schnittstelle zur Verfügung stellt. Tapestry kommuniziert über JavaServer Pages (JSP) und Servlets direkt mit der Prozessschicht und der Serviceschicht. Die Prozessschicht (2) benutzt jBPM (siehe Kapitel 3.5) um die Prozesse abzubilden. Diese wiederum erhalten, genau wie die Präsentationsschicht, alle Daten von der Serviceschicht. Die Serviceschicht (3) kapselt jegliche Kommunikation mit dem Modell und ist über Session Beans (siehe Kapitel 3.2.2) zugreifbar. Intern greift sie mit Entity Beans (siehe Kapitel 3.2.3) auf die Datenbankschicht zu. Die Datenbankschicht (4) besteht aus einer relationalen Datenbank (HSQLDB, siehe [The hsql Development Group]), in welcher das Modell gespeichert wird.

## 5.2. Einrichtung

Zu Beginn der Projektgruppe wurde die entsprechende Entwicklungsumgebung für die Umsetzung des Projekts aufgebaut. Dazu gehört die Auswahl eines Entwicklungstools, eines Build Tools, eines Application Servers, sowie einer Datenbank und einem Tool zum Erstellen von Prozessdesigns. Die folgenden Abschnitte befassen sich mit diesem Thema.

In der ersten Sitzung der Projektgruppe wurden mehrere Kleingruppen gebildet, um die möglichen Tools zu testen. In den meisten Fällen wurden nur zwei Alternativen getestet, da die Testphase nur eine Woche lang war. Es war den Teilnehmern sehr wichtig, schnell eine Entscheidung über die Tools zu treffen, damit sie mit dem ersten Projekt anfangen konnten. Basierend auf den Ergebnissen der Kleingruppen wurde über die verschiedenen Tools diskutiert und abgestimmt.

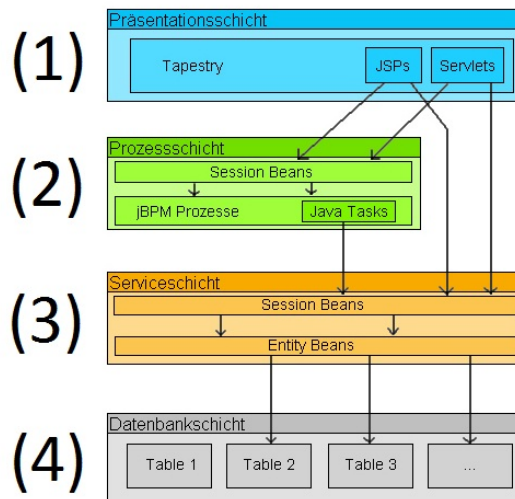


Abbildung 5.1.: Architektur - Schichtenmodell

Beim Aufbau der Entwicklungsumgebung standen als Entwicklungstools hauptsächlich Eclipse und Netbeans zur Auswahl. Eclipse und Netbeans bieten eine große Anzahl an Plugins. Da die meisten Teilnehmer bereits mit Eclipse vertraut waren und die Vor- und Nachteile beider Tools sich die Waage hielten, ist die Entscheidung auf Eclipse gefallen.

Als Datenbank hat sich die Projektgruppe auf Empfehlung unserer Betreuer für die HyperSQL Datenbank (HSQLDB) entschieden, da die Datenmengen, die im Rahmen der Projektgruppe bewältigt werden müssen, sich im Rahmen halten und die HSQLDB deshalb den Anforderungen genügt. In der Diskussion vor der Entscheidung wurde als Alternative Postgres-SQL vorgeschlagen. Postgres ist ein sehr robustes Relational Database Management System (RDBMS) und bietet sehr viele Funktionalitäten. Allerdings ist die Verwendung als Speicher-Datenbank nicht möglich. Letztendlich war Postgres-SQL etwas zu groß für unsere HelloWorld-Anwendung. Es ist aber nicht ausgeschlossen, dass wir drauf zurückgreifen, falls die Anforderungen im Laufe der Projektgruppe wachsen.

Als Repository für unser Projekt standen Origo und Sourceforge zur Wahl. Origo bietet nicht nur alles unter einem Dach, sondern auch eine sehr schnelle Freischaltung des Projekt-Repository. Die Check-in und Check-out sind schneller als bei Sourceforge und der Code muss nicht quelloffen sein. Die Gruppe entschied sich daher für Origo.

Die zwei bekanntesten Build-Tools sind Maven und Ant. In der Seminarphase der Projektgruppe haben wir uns nur mit Maven beschäftigt. Im Vergleich zu Ant bietet es viele Vorteile. Man spart z.B. viel Zeit bei der Generierung des Build-Skript und es gibt viele vordefinierte Goals. Da wir durch die Seminarphase bereits mit Maven vertraut waren, haben wir dies als Build-Tool gewählt.

jBPM ist ein freies von jBoss entwickeltes Business Process Management System

(siehe Kapitel 3.5). jBPM wurde für das Prozessdesign gewählt, da es quelloffen ist, und für Eclipse ein Plugin existiert. Diese Wahl wurde zudem von der Tatsache bekräftigt, dass wir kleine Beispiele ausführen konnten, um uns von der Einfachheit von jBPM zu überzeugen.

Bei der Wahl des Application Servers mussten wir uns zwischen JBoss und Glassfish entscheiden. Nachdem wir beide getestet hatten, fiel die Entscheidung auf JBoss, da wir darauf jBPM und Tapestry deployen konnten. Auf Glassfish ist es uns nicht gelungen jBPM erfolgreich zu deployen.

Als Webdesign-Framework standen Tapestry und Wicket zur Verfügung. Beide Frameworks ermöglichen eine saubere Trennung von HTML und Java und arbeiten nach dem MVC-Prinzip. Bei Wicket muss man die Java-Klassen und Templates immer synchron halten, was bei Tapestry nicht der Fall ist. Die Teilnehmer hatten darüberhinaus den Eindruck sich schnell in Tapestry einarbeiten zu können. Darum wurde Tapestry von der Gruppe gewählt. Bei der Einrichtung der Entwicklungsumgebung sind ein paar Probleme aufgetreten. Zum Einen gehörte das Deployen der Tapestry-HelloWorld Anwendung auf dem JBoss. Als Lösung wurde der ClasspathURLConverterJBoss5 von apache verwendet. Damit funktionierte das Deployen problemlos. Zum Anderen gab es noch das Problem, dass der Zugriff auf die Datenbank nicht möglich war. Um dieses Problem zu beheben wurde auf dem JBoss eine Datasource-Definition hinzugefügt, sodass der Zugriff wieder möglich war.

### 5.3. Testen

Zur Gewährleistung eines agilen Software-Entwicklungsprozesses wurden die Teilprojekte der Service- und Prozessschicht stets mit Hilfe von Unit-Tests auf Korrektheit überprüft. Da die Präsentationsschicht in unseren Projekten lediglich die Funktionalitäten der darunter liegenden Schichten verwendet und selbst keinerlei Geschäftslogik implementiert, wurde in dieser Schicht auf automatisierte Tests verzichtet. In unserem Fall musste die Webschicht ausschließlich auf das korrekte Rendern der Templates und eine korrekte Verlinkung der Seiten untereinander getestet werden. Diese Bereiche lassen sich jedoch nur mit einem unverhältnismäßig großen Aufwand automatisiert testen. So haben sich zwischendurch die Link-Pfade und Templates, insbesondere in den ersten beiden Projekten, häufig geändert. Die Tests hätten ständig an die neuen Begebenheiten angepasst werden müssen, wohingegen eine manuelle Überprüfung binnen kürzester Zeit möglich war.



# 6. Projekte

Innerhalb des ersten Semesters der Projektgruppe wurden zwei Projekte bearbeitet, die im Folgenden genauer erläutert werden. Hierbei soll jeweils auf die Aufgabenstellung, die Herangehensweise, den Ablauf und die dabei aufgetretenen Probleme eingegangen werden.

## 6.1. Projekt 1: Lagerverwaltung

Beim ersten Projekt wurde die Architektur des Schichtenmodells, bestehend aus vier Schichten (s. Abschnitt 5.1) eingesetzt. Zum Einsatz sind die Datenbankschicht, die Serviceschicht, die Prozessschicht und die Webschicht gekommen, welche untereinander interagieren. Im Abschnitt 6.1.2 werden diese Schichten in Bezug auf die Lagerverwaltung näher betrachtet.

### 6.1.1. Beschreibung

In dem ersten Projekt wurde der Projektgruppe die Aufgabe gestellt eine funktionierende Lagerverwaltung zu implementieren. In dem Lager soll es Regale geben in dem Paletten eingelagert werden. Die Regalplätze sind durchnummeriert. Es gibt 10 Regale, die je 20 Paletten breit und 4 Paletten hoch sind, und die Paletten haben eine Identifikationsnummer, die die Artikel darauf widerspiegeln. Weiterhin gibt es eine Mindestmenge von Paletten eines bestimmten Artikels im Lager, bei dessen unterschreiten eine Mail geschickt werden soll. Insgesamt soll es einen Administrator, einen Mitarbeiter und einen Gabelstaplerfahrer als Akteure geben, die das Lager verwalten können.

In dem Lager sollen bestimmte Funktionen möglich sein, wie die Funktion „Einlagern“. Ein Mitarbeiter oder Administrator soll Artikel ins System einlagern können. Dabei sucht er einen bekannten Artikel aus, oder gibt einen neuen Artikel in das System ein. Dann wählt er die Anzahl und lagert sie in das System ein. Der Artikel erscheint anschließend auf der Auftragsliste der Gabelstaplerfahrer mit einem Lagerplatz.

Als nächste Funktion soll das Auslagern funktionieren, wobei ein Mitarbeiter oder Administrator Artikel aus dem System auslagern können soll. Dabei wählt er die Palette mit dem gewünschten Artikel aus und lagert sie aus dem System aus. Der Artikel erscheint dann auf der Auftragsliste der Gabelstaplerfahrer, die für das Ein- und Auslagern der Paletten zuständig sind.

Weiterhin soll ein Gabelstaplerfahrer oder Administrator die Auftragsliste der Gabelstaplerfahrer einsehen können und einen Auftrag darauf als „wird bearbeitet“ markieren können. Außerdem soll er einen als „wird bearbeitet“ markierten Auftrag als erledigt markieren können. Wenn es sich um einen Einlagern-Auftrag handelt, so wird dem Lager die Palette zugeordnet. Der Lagerplatz soll dabei nach einer bestimmten Einlagerungsstrategie erfolgen. Bei einem Auslagern-Auftrag wird die Palette aus dem Lager entfernt und ins Packlager geladen.

Ein Mitarbeiter oder Administrator soll eine Palette aus dem Packlager entweder als verbraucht oder als zurücklagern markieren können. Wird eine Palette des Packlagers als verbraucht markiert, so wird sie aus dem Packlager entfernt. Unterschreitet dabei das Lager (zuzüglich Packlager) die Mindestanzahl des Artikels, so wird eine Mail an eine verwaltende Stelle geschickt. Wird sie als zurücklagern markiert, dann taucht sie wieder auf der Auftragsliste der Gabelstaplerfahrer mit einem Lagerplatz auf.

Ein Administrator soll jederzeit die Lagerplätze des Lagers und Depotlagers einsehen, und (zum Zwecke der Korrektur) die Plätze editieren können. Dabei soll es möglich sein, jeden Platz mit einer Palette zu belegen, bzw. von jedem Platz die darauf lagernde Palette zu entfernen.

Schließlich soll ein Mitarbeiter oder Administrator jederzeit das Lager einsehen können. Dabei soll es möglich sein, bei jedem Platz einzusehen welcher Artikel sich dort befindet. Außerdem soll eine Gesamtauslastung des Lagers einsehbar sein.

Beim Einlagern oder Zurücklagern von Paletten ins Lager soll das System einen Lagerplatz für die Palette berechnen. Dabei soll eine chaotische Lagerhaltung gewählt werden, was bedeutet, dass der Lagerplatz zufällig gewählt wird. Wichtig ist nur, dass der Platz frei ist. Die Strategie soll allerdings leicht austauschbar und durch klügere Strategien ersetzt werden können.

## 6.1.2. Ablauf

Zur Umsetzung des ersten Projekts (s. Abschnitt 6.1.1) wurde zuerst eine Anforderungsanalyse durchgeführt, indem User Stories erstellt und priorisiert wurden. Dadurch wurde verdeutlicht was der User genau tun möchte und was für die Lagerverwaltung notwendig sei. Folgende User Stories mit den zugehörigen Prioritäten wurden erstellt.

### User Stories mit Priorität 1

1. **Auslagern:** Wenn Lagermitarbeiter in der Packstation Artikel aus dem Lager brauchen, sollen sie Artikel und Menge im System angeben können. Entsprechende Aufträge, einer oder mehrere, für die Staplerfahrer sollen generiert werden.
2. **Einlagern:** Wenn neue Ware am Wareneingang ankommt, soll ein Lagermitarbeiter dies dem System mitteilen können. Ein Lagerplatz, oder mehrere La-

gerplätze, werden heraus gesucht und entsprechende Aufträge für die Staplerfahrer generiert.

### **User Stories mit Priorität 2**

1. **Artikel registrieren:** Ein Mitarbeiter soll neue Artikeltypen dem System bekanntmachen können.
2. **Bestand des Lagers anzeigen:** Ein Mitarbeiter soll sich zu jeder Zeit den aktuellen Bestand des Lagers anzeigen lassen können um Informationen über Lagerplätze und deren Auslastung zu erhalten.

### **User Stories mit Priorität 3**

1. **Administrieren des Lagers:** Ein Administrator soll manuell die Lagerbestände editieren können, um das Lager zu korrigieren und zu initialisieren.

### **User Stories mit Priorität 4**

1. **Palette als leer markieren:** Ein Lagermitarbeiter soll eine Palette als leer markieren können. Der zugehörige Lagerplatz der Palette soll dann frei gegeben werden. Die Palette muss zurück zum Lagerplatz.
2. **Palette zurücklagern:** Ein Lagermitarbeiter soll eine Palette als "nicht mehr im Gebrauch" markieren können. Die Palette mit den Artikeln muss zurück zum Lagerplatz.

Im Nachhinein konnten aus den User Stories konkrete Arbeitsschritte (Tasks) abgeleitet werden, welche einen detaillierten Überblick für die Implementierung anbieten konnten. Die entstandenen Arbeitsschritte wurden den vier Schichten (Web-schicht, Prozessschicht, Serviceschicht, Datenbankschicht) zugeordnet. Im folgenden sind die einzelnen Tasks aufgelistet und beschrieben.

### **Tasks mit Priorität 1**

1. **Datenbankmodelle erstellen:** Es muss das Datenbank-Modell erstellt werden welches das Lager und die Artikel abbildet.
2. **Oberfläche zum Einlagern:** Es muss eine Oberfläche erstellt werden die es ermöglicht einen Artikel einzulagern.
3. **Oberfläche zum Auslagern:** Es muss eine Oberfläche erstellt werden die es ermöglicht einen Artikel auszulagern.
4. **Services zum Ein- und Auslagern:** Es müssen Services erstellt werden die Ein- und Auslagern durchführen.

## Tasks mit Priorität 2

1. **Oberfläche zum Artikel registrieren:** Es muss eine Oberfläche erstellt werden mit der Artikel registriert werden können.
2. **Service zum Artikel registrieren:** Es muss ein Service erstellt werden mit der Artikel registriert werden können.
3. **Oberfläche zum Bestand anzeigen:** Es muss eine Oberfläche erstellt werden mit der der Lagerbestand angezeigt wird.
4. **Service zum Bestand anzeigen:** Es muss ein Service erstellt werden mit dem der Bestand abgefragt wird.

## Tasks mit Priorität 3

1. **Oberfläche zum Administrieren des Lagers:** Es muss eine Oberfläche erstellt werden mit der das Lager administriert werden kann.
2. **Service zum Administrieren:** Es muss ein Service erstellt werden mit dem das Lager administriert werden kann.

## Tasks mit Priorität 4

1. **Oberfläche zum Packlager verwalten:** Es muss eine Oberfläche erstellt werden mit der die Paletten im Packlager zurückgelagert oder als leer markiert werden kann.
2. **Service zum Packlager verwalten:** Es muss ein Service erstellt werden mit dem Paletten im Packlager verwaltet werden können.

Im Folgenden wird beschrieben wie die oben genannten vier Schichten in der Lagerverwaltung eingesetzt worden sind.

**Prozessschicht** Die Projektgruppe hat sich entschieden einen palettenbasierten Prozess zu definieren (siehe Abbildung 6.1), um den Zyklus einer Palette im Lager darzustellen. Wenn eine Palette nicht leer ist, kann sie auf einen leeren Lagerplatz eingelagert werden. Wenn aus einer Palette Artikel benötigt werden, dann wird die jeweilige Palette aus dem Lager entnommen und ins Depot abgestellt. Im Depot ist es möglich aus der Palette Artikel zu entnehmen. Wenn die Palette nach der Entnahme keine Artikel mehr enthält, so wird der Prozess für diese Palette beendet. Andernfalls wird die Palette zurückgelagert.



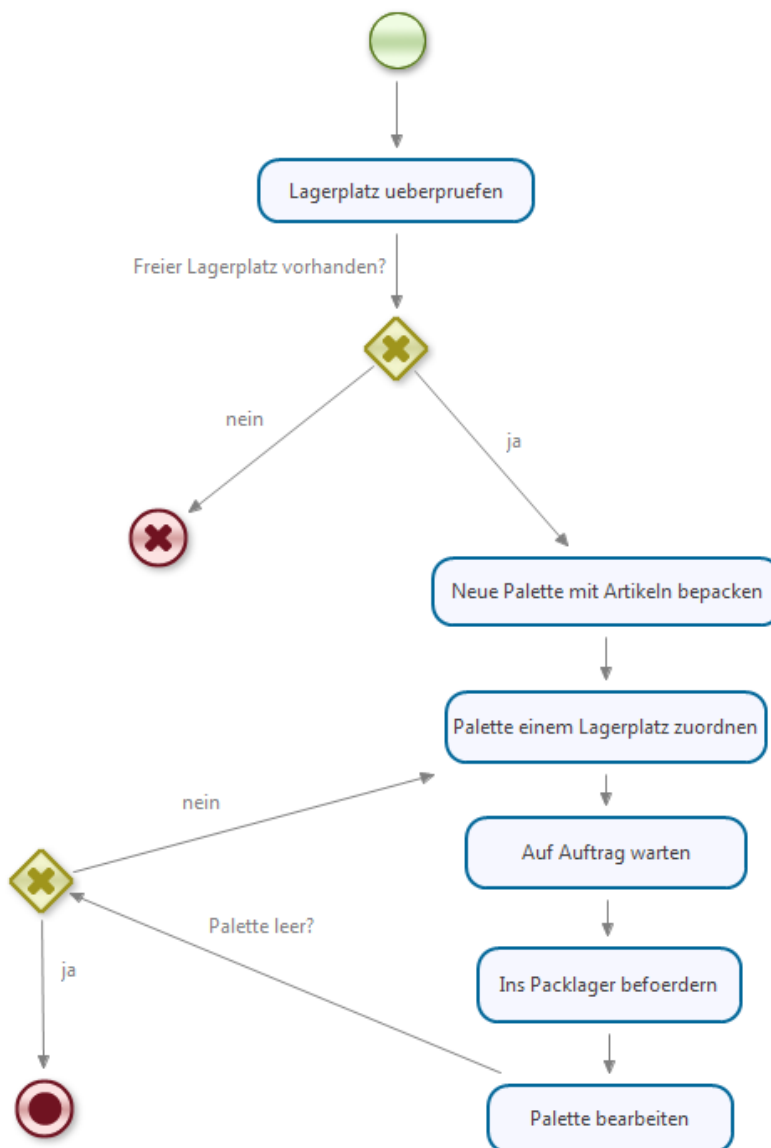


Abbildung 6.1.: Palettenprozess

**Serviceschicht** In der Serviceschicht wurde eine Einlagerungsstrategie eingesetzt, die dafür zuständig ist, alle freien Lagerplätze auszugeben. Dazu werden die zugehörigen Daten aus der Datenbankschicht entnommen. Wenn ein Lagerplatz besetzt bzw. frei wird, bekommt die Datenbank unmittelbar den aktuellen Zustand der Paletten und der Artikel, und die Tabellen werden aktualisiert.

**Datenbankschicht** In der Datenbank wurden die drei Tabellen Article, Pallet und Tray angelegt (siehe Abbildung 6.2). Ein Artikel hat einen Namen(name), eine Artikelnummer(ID) und eine Mindestmenge (minAmount). Eine Palette (pallet) ist durch eine Palettennummer(ID), eine Artikelnummer(articleID) und durch den Ort an dem sie sich gerade befindet (packingDepot), definiert. Eine Palette kann sich entweder im Depot oder an einem Lagerplatz befinden. Das Lager besitzt die Tabelle Tray, woraus die Lagerfächer ermittelt werden können. Jedes Fach hat eine Palettennummer (palletID) und die Koordinaten (rackNr, parcelNr, height), durch die jedes Fach identifiziert ist.

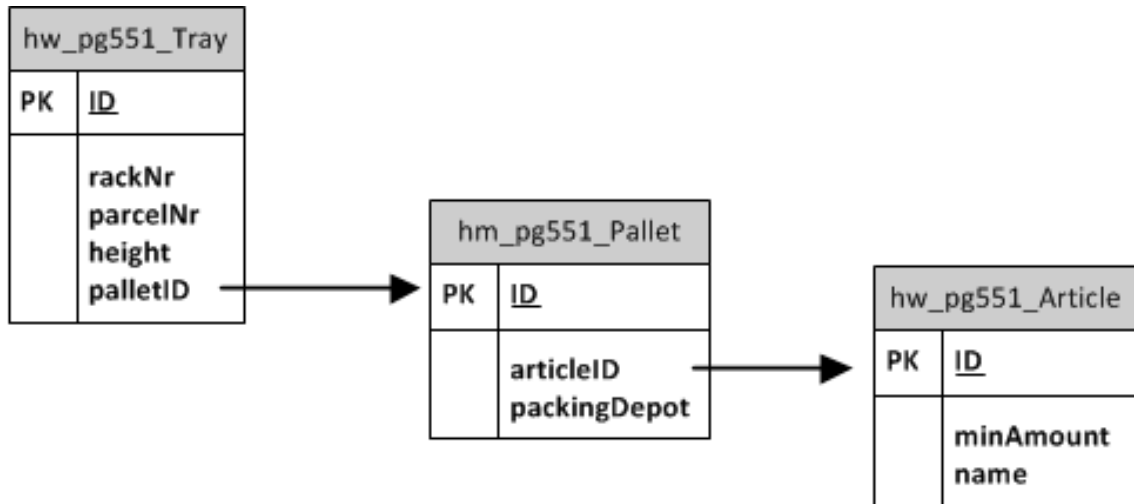


Abbildung 6.2.: Datenmodell Projekt 1

**Webschicht** Nachdem die Implementierung abgeschlossen war, waren die aus den Tasks geforderten Oberflächen in der Webschicht entstanden. Die beiden Funktionen der Administration und der Verwaltung des Packlagers wurden nach Absprache mit dem Kunden nicht in das Programm integriert.

- Einlagern (s. Abbildung 6.3)  
Hierbei kann der Benutzer sowohl den Artikel auswählen, der sich auf der Palette befindet, als auch die Anzahl der einzulagernden Paletten eingeben. Nachdem er seine Eingaben an das System geschickt hat, wird die Gesamtanzahl an Paletten eines Artikels angezeigt.
- Auslagern
- Artikel Registrieren
- Bestand anzeigen (s. Abbildung 6.4)  
Wie in Abbildung 6.4 zu sehen ist, kann sich der Benutzer den Bestand seines

Lagers anzeigen lassen. Hierbei sind die Artikel nach ArtikelId nummeriert und mit ihrem Namen angegeben. Auf der rechten Seite wird die Anzahl der Paletten mit den zugehörigen Artikeln ausgegeben.

Index Articles Stock Unstock ProcessTest ArticlesTest

## Paletten einlagern

Artikel:

Anzahl einzulagernder Paletten:

Es wurden 4 Paletten des Artikels 1 eingelagert.

Abbildung 6.3.: Einlagern

Die Anforderungen waren hiermit erfüllt, wobei vom Kunden ein zusätzliches Feature erwünscht war. In der Lagerverwaltung wurde eine Benutzerauthorisierung mit Java Authentication and Authorization Service (JAAS) anhand der bestehenden Implementation des JBoss implementiert.

### 6.1.3. Zusammenfassung

Zusammenfassend kann man erwähnen, dass die Lagerverwaltung zufriedenstellend nach dem Schichtenmodell implementiert worden ist. Das Programm war lauffähig und erfüllte die geforderten und von der Projektgruppe gesetzten Ziele (s. Abschnitt 2.2).

Aufgrund von auftretenden Fehlern, die behoben werden mussten und die nicht aus dem Blick gelassen werden durften, hat sich die Projektgruppe dafür entschieden, den Pivotaltracker (s. Kapitel 4.3) als Bugtracker einzusetzen. Dieser Bugtracker sollte u.a. die Kommunikation innerhalb der Gruppe verbessern, indem man jedem Mitglied Aufgaben zuweist und jeder sich im Klaren darüber ist welche Aufgaben von welchem Mitglied bearbeitet werden. Dies soll dazu führen, dass Aufgaben nicht doppelt bearbeitet werden. Zusätzlich kann man erkennen wer welche Aufgaben bearbeitet hat.

[Index](#)
[Articles](#)
[Stock](#)
[Unstock](#)
[ProcessTest](#)
[ArticlesTest](#)

## Artikel-Liste





 Id	Name	Min. Paletten
 1	Schrauben	3
 2	Muttern	3
 3	Klemmen	2

Abbildung 6.4.: Bestand anzeigen

Die Teilnehmer haben sich in den allgemeinen Ablauf und die Organisation von Projekten eingefunden und können die nächsten Projekte somit strukturierter angehen. Zudem wurde die Entwicklungsumgebung aufgebaut, so dass der Arbeitsaufwand hierfür bei den nächsten Projekten entfällt.

Die fertige Lagerverwaltung wurde bis auf zwei fehlende Funktionen zum vereinbarten Abgabetermin dem Kunden präsentiert, womit dieses Projekt als abgeschlossen erklärt werden konnte.

## 6.2. Projekt 2: Stundenzettel

Im zweiten Projekt ging es darum, eine Verwaltung von Arbeitszeiten anwenderfreundlicher zu gestalten. Wie schon im ersten Projekt wurde auch in diesem die Architektur des in Abschnitt 5.1 beschriebenen Schichtenmodells eingesetzt. Auf die Verwendung der einzelnen Schichten innerhalb dieses Projekts wird im Abschnitt 6.2.2 eingegangen.

### 6.2.1. Beschreibung

#### 6.2.1.1. Aufgabenstellung

Im zweiten Projekt wurde die Projektgruppe vor die Aufgabe gestellt, eine bestehende Excel-Datei zur Kalkulation und Verwaltung von monatlichen Arbeitszeiten mit Hilfe eines Webinterfaces bearbeiten zu können. Hierbei stand im Fokus, von der Komplexität der Formeln in den Zellen der Tabelle zu abstrahieren und so un-

abhängig von der Verständlichkeit der Formeln - die unter Umständen nur für den ursprünglichen Programmierer der Tabelle verständlich sind - einen einfachen Umgang mit der Tabelle zu gewährleisten. Ziel dieses Projekts war es, sich in den Umgang mit Excel-Tabellen in Hinsicht auf Auslesen, Manipulieren und Einpflegen von Werten in Zellen einzuarbeiten und sich mit den dafür benötigten Tools vertraut zu machen, um schlussendlich für das dritte Projekt, in dem es voraussichtlich um die benutzerfreundliche Bearbeitung einer weitaus komplexeren Excel-Tabelle gehen wird, eine Wissensgrundlage innerhalb der Projektgruppe zu schaffen.

### **6.2.1.2. Beschreibung der Excel-Datei**

Das Excelsheet (s. Abbildungen 6.5 und 6.6) besteht aus einem Beispielblatt zu Beginn, gefolgt von je einem Blatt pro Monat. Ein Monatsblatt umfasst allgemeine Informationen zum Mitarbeiter, die im Kopf jedes Monatsblatts erneut aufgeführt werden, und alle Tage des Monats mit entsprechendem Wochentag und die Arbeitszeiten je Projekt in tabellarischer Form. Dabei sind Feiertage von vornherein grau markiert. Die Mitarbeiterdaten bestehen aus

- Name,
- Vorname,
- Besoldungs- bzw. Entgeltgruppe,
- Funktion,
- LBV-Personalnummer,
- Wochenarbeitszeit in Prozent und
- der Wochenarbeitszeit in Stunden, sowie
- einem Feld für die Unterschrift des Mitarbeiters.

Zudem befindet sich auf jedem Datenblatt links neben den Mitarbeiterdaten eine kurze Beschreibung zum Stundenzettel, rechts davon das Logo der TU Dortmund und am unteren Ende des Datenblatts eine Legende zu den Symbolen. In der tabellarischen Übersicht über die Arbeitszeiten ist pro Spalte ein Tag des Monats mit Datum gelistet, sodass eine Übersicht über den gesamten Monat entsteht. Pro Zeile können dann die Arbeitszeiten für verschiedene Projekte eingegeben werden, je in einer Zeile pro Projekt. Die Projekte werden nicht namentlich sondern durch eine Projektkennzahl bzw. Haushaltskennzahl angeführt. Urlaub wird beim Eintragen der Arbeitszeiten durch ein gelb hinterlegtes U angezeigt. In der letzten Zeile werden die Arbeitszeiten pro Tag aufaddiert und angezeigt. Die beiden letzten Spalten der Tabelle geben schließlich die Gesamtstunden pro Projekt an, in der die Arbeitszeiten pro Projekt über den Monat hin aufaddiert werden, und bieten Platz für die Unterschriften der jeweiligen Projektleiter.



19	20	21	22	23	24	25	26	27	28	29	30	31	Gesamtstd .pro Projekt	Unterschrift Projektleiter/in
Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So		
		2,5	8,0			8,0	8,0	4,0		8,0			89,83	
8,0	8,0												25,00	
		5,0											14,50	
								4,0	8,5				14,50	
													0,00	
													0,00	
													0,00	
													0,00	
													0,00	
													0,00	
8,00	8,00	7,50	8,00	0,00	0,00	8,00	8,00	8,00	8,50	8,00	0,00	0,00	143,83	

Abbildung 6.6.: Beispielabbildung des Excelsheets Teil 2

## 2. Datenhaltung: Stunden, Projekte, Benutzer

Die erste Tabelle der Datei beinhaltet immer ein Beispiel. Die Stammdaten, die im Kopf jedes Tabellenblattes einzutragen sind, sollen pro Benutzer einmal eingetragen werden können. Um ein Projekt anzulegen, sind Informationen über die Kennzahl, den Namen (frei vom Benutzer wählbar), die maximale Monatsarbeitszeit des Benutzers in Stunden und den Projektleiter anzugeben. Korrekturen der eingegeben Daten sollen jederzeit möglich sein.

Es gibt vier verschiedene Zeitkontingente:

- Maximale Arbeitszeit pro Tag: Beträgt immer 10 Stunden
- Wochenarbeitszeit: Tarifabhängig, erhält man aus den Stammdaten des Benutzers
- Maximale Arbeitszeit pro Projekt im Monat
- Monatliche Gesamtarbeitszeit

### 3. Features

Stunden sollen zunächst ausschließlich manuell vom Benutzer eingetragen werden können. Später soll dies auch per „Live-Modus“ möglich sein. Der Benutzer soll bestimmen können, ob die verbleibende reguläre Arbeitszeit eines Tages / eines Monats automatisch für ein zuvor von ihm bestimmtes „Default-Projekt“ eingetragen wird, oder nicht.

### 4. Warnungen

Es soll eine Warnung eingeblendet werden, wenn die maximale Tagesarbeitszeit überschritten wurde, sowie wenn am Ende eines Monats die maximale monatliche Arbeitszeit insgesamt oder innerhalb eines Projekts überschritten wurde. Das Eintragen von Arbeitszeiten für Samstage und Sonntage soll möglich sein. Allerdings soll anschließend eine Warnung erscheinen. Der Benutzer soll seine bevorzugte Art von Warnungen und Benachrichtigungen einstellen können. Hierbei hat er die Wahl zwischen Warnungen per E-Mail und Warnungen innerhalb der Oberfläche.

### 5. Abgaben, Excel Download

Die Abgabe der Stundenzettel erfolgt monatlich. Dazu wird das abgeschlossene Tabellenblatt des jeweiligen Monats als Excel-Tabelle ausgegeben und kann zur Unterschrift an die Projektleiter weitergereicht werden.

## 6.2.2. Ablauf

### 6.2.2.1. User Stories

Nachdem die Anforderungsanalyse im Gespräch mit dem Kunden stattgefunden hat und dokumentiert worden ist, wurden hieraus - wie schon im ersten Projekt - User Stories entwickelt, die anschließend vom Kunden priorisiert wurden. Diese werden im Folgenden aufgelistet.

#### User Stories mit Priorität 1

1. **Anmeldung:** Der Benutzer möchte sich bei der Anwendung anmelden können.
2. **Registrierung:** Der Benutzer möchte sich bei der Anwendung registrieren können.



3. **Neues Projekt anlegen:** Der Benutzer möchte der Anwendung ein neues Projekt bekanntmachen, für das er dann Stunden eintragen kann.
4. **Arbeitszeit eintragen:** Der Benutzer möchte für einen bestimmten Tag und ein bestimmtes Projekt eine bestimmte Anzahl von Stunden und Minuten eintragen, die er für dieses Projekt gearbeitet hat.
5. **Arbeitszeiten anzeigen:** Der Benutzer möchte sich für einen bestimmten Monat anzeigen lassen, welche Arbeitszeiten er bereits eingetragen hat.
6. **Arbeitszeiten ändern:** Der Benutzer möchte bereits eingetragene Arbeitszeit entfernen und ändern können.
7. **Urlaub eintragen:** Der Benutzer möchte eintragen, dass er an einem bestimmten Tag Urlaub hatte.
8. **Excel-Export:** Der Benutzer möchte alle eingetragenen Zeiten eines Monats in eine Excel-Datei exportieren und ausdrucken, die er dann von den Projektleitern unterschreiben lässt und bei der Verwaltung abgibt. Es steht eine entsprechende Vorlage dafür zur Verfügung.
9. **Abmelden:** Der Benutzer möchte sich von der Anwendung abmelden können.

## User Stories mit Priorität 2

1. **Warnungen:** Der Benutzer möchte von der Anwendung benachrichtigt/gewarnt werden, wenn
  - die von ihm normalerweise zu leistende Arbeitszeit im Monat (alle Projekte) unter- oder überschritten wurde.
  - die von ihm in diesem Monat zu leistende Arbeitszeit für ein bestimmtes Projekt überschritten wurde bzw. am Ende eines Monats immer noch unterschritten wird
  - er für einen bestimmten Tag mehr als 10 Arbeitsstunden eingetragen hat.
  - er Arbeitsstunden für einen Samstag, Sonntag oder Feiertag eingetragen hat.
2. **Live-Modus:** Der Benutzer möchte es der Anwendung mitteilen, wenn er anfängt, an einem bestimmten Projekt zu arbeiten und auch, wenn er damit wieder aufhört. Die Anwendung trägt dann die zwischen Start und Ende vergangene Zeit automatisch für den aktuellen Tag und das gewählte Projekt ein.

3. **Automatische Verteilung von Restarbeitszeit:** Die Anwendung soll auf Wunsch des Benutzers automatisch die nicht eingetragene Restarbeitszeit eines Tages/ eines Monats bei einem Default-Projekt eintragen. Dies geschieht entweder am Ende eines Arbeitstages oder am Ende eines Monats. Dafür soll der Benutzer ein der Anwendung bekanntes Projekt als Default-Projekt festlegen können. Diese Funktion soll ein- und ausschaltbar sein.

### User Stories mit Priorität 3

1. **Optionen für Benachrichtigungen:** Der Benutzer möchte wählen, wie er von der Anwendung benachrichtigt/gewarnt werden möchte:
  - Per E-Mail
  - Durch die Weboberfläche
  - Beides
2. **Projektleiter als weiterer Benutzer:** Der Projektleiter möchte sich bei der Anwendung an-/und abmelden, sowie registrieren können.
3. **Unterschrift durch Projektleiter:** Der Projektleiter möchte der Anwendung mitteilen, dass er den Stundenzettel eines bestimmten Mitarbeiters für einen bestimmten Monat unterschrieben hat.

Gleichzeitig zum Bestimmen der User Stories wurde nach Tools recherchiert, die Java-APIs zum Lesen und Schreiben von Excel-Dateien bereitstellen. Hier wurde der Apache POI näher betrachtet und wegen seiner einfachen Bedienung für die Umsetzung des Projekts verwendet.

Nachdem die Excel-Datei genau analysiert, die UserStories dokumentiert und priorisiert und APIs zur Manipulation von Excel-Dokumenten recherchiert worden waren, ging es anschließend darum, mögliche Prozesse zu definieren. Es wurde überlegt, wie eine Oberfläche zum Eintragen von Arbeitsstunden aussehen könnte und erste Überlegungen zum Datenmodell angestellt.

Die einzelnen Überlegungen werden im Folgenden ihren Schichten entsprechend erläutert:

#### 6.2.2.2. Prozessschicht

Für die Prozessschicht hat sich die Projektgruppe auf die Verwendung von vier Prozessen geeinigt.

**Täglicher Prozess** Es gibt einen täglichen Prozess, der am Ende jedes Tages die eingetragenen Stunden überprüft, falls Warnungen auftreten eine Hinweis-E-Mail verschickt, und nicht verbuchte Stunden auf das Defaultprojekt bucht (siehe Abbildung 6.7). Nach dem Start befindet sich der Prozess im Wartezustand bis das

Ende des Tages erreicht ist. Anschließend werden die verbleibenden Stunden zum Defaultprojekt hinzugefügt. Falls eine Benachrichtigung per E-Mail aktiviert wurde, wird diese verschickt. Danach wird das nächste Fälligkeitsdatum berechnet und der tägliche Prozess geht wieder in den Wartezustand.

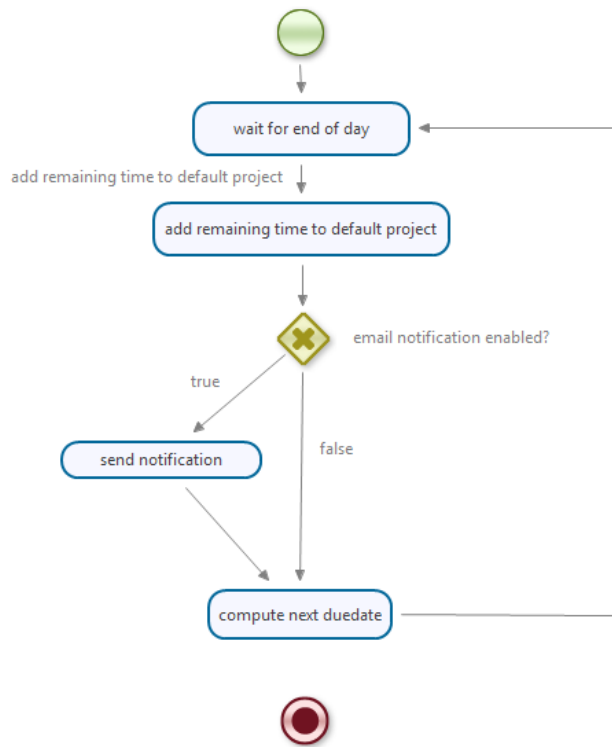


Abbildung 6.7.: Täglicher Prozess

**Wöchentlicher Prozess** Desweiteren gibt es einen wöchentlichen Prozess, der am Ende jeder Woche die eingetragenen Stunden überprüft, mit der Wochenarbeitszeit vergleicht und ggf. Hinweis-E-Mails verschickt (siehe Abbildung 6.8). Nach dem Start befindet sich der Prozess im Wartezustand bis das Ende der Woche erreicht ist. Wenn die Woche zu Ende ist, werden die verbleibende Stunden zum Defaultprojekt hinzugefügt. Sofern eine Benachrichtigung per E-Mail eingestellt wurde, wird diese verschickt. Danach wird das nächste Fälligkeitsdatum berechnet und der wöchentliche Prozess geht wieder in den Wartezustand.

**Monatlicher Prozess** Ein monatlicher Prozess überprüft am Ende eines Monats die eingetragenen Stunden und verschickt ggf. Hinweis- und Erinnerungsmails zum Schließen des Monats (siehe Abbildung 6.9). Anfangs befindet sich der Prozess in einem Wartezustand (wait for end of month). Wenn ein Monat zu Ende geht, wird zuerst der nächste monatliche Prozess gestartet. Anschließend werden, falls erwünscht,

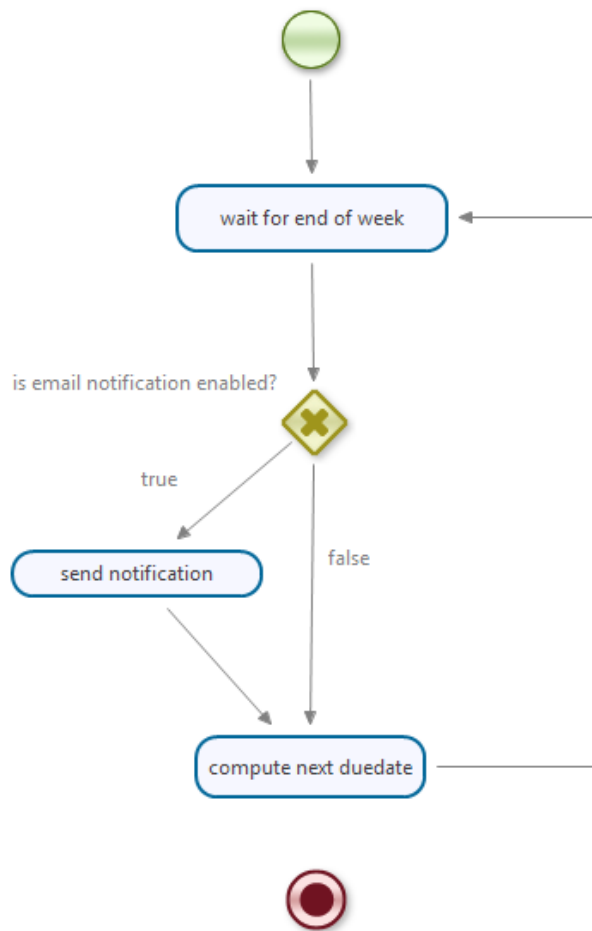


Abbildung 6.8.: Wöchentlicher Prozess

die Benachrichtigungen verschickt. Der Prozess wartet dann auf das Beenden des Monats. Bei Timeout werden die Benachrichtigungen erneut verschickt und der Prozess befindet sich wieder im Wartezustand auf das Beenden des Monats. Beim Beenden durch den Benutzer ist der Prozess gestoppt.

**LiveModus Prozess** Ein LiveModus-Prozess trägt nach Beenden durch den Benutzer oder nach automatischem Stoppen am Ende des Tages die Stunden ein und verschickt ggf. Hinweis-E-Mails (siehe Abbildung 6.10). Beim Starten des LiveModus durch den Benutzer in der Weboberfläche wird intern der LiveModus-Prozess gestartet. Er befindet sich initial in einem Wartezustand bis er durch den Benutzer beendet wird, oder das Ende des Tages erreicht wird, was zu einem Timeout führt. Im Falle eines Timeouts wird, falls eingestellt, eine Hinweis-E-Mail an den Benutzer verschickt.

Es werden also regelmäßig die Eintragungen der Arbeitsstunden überprüft und

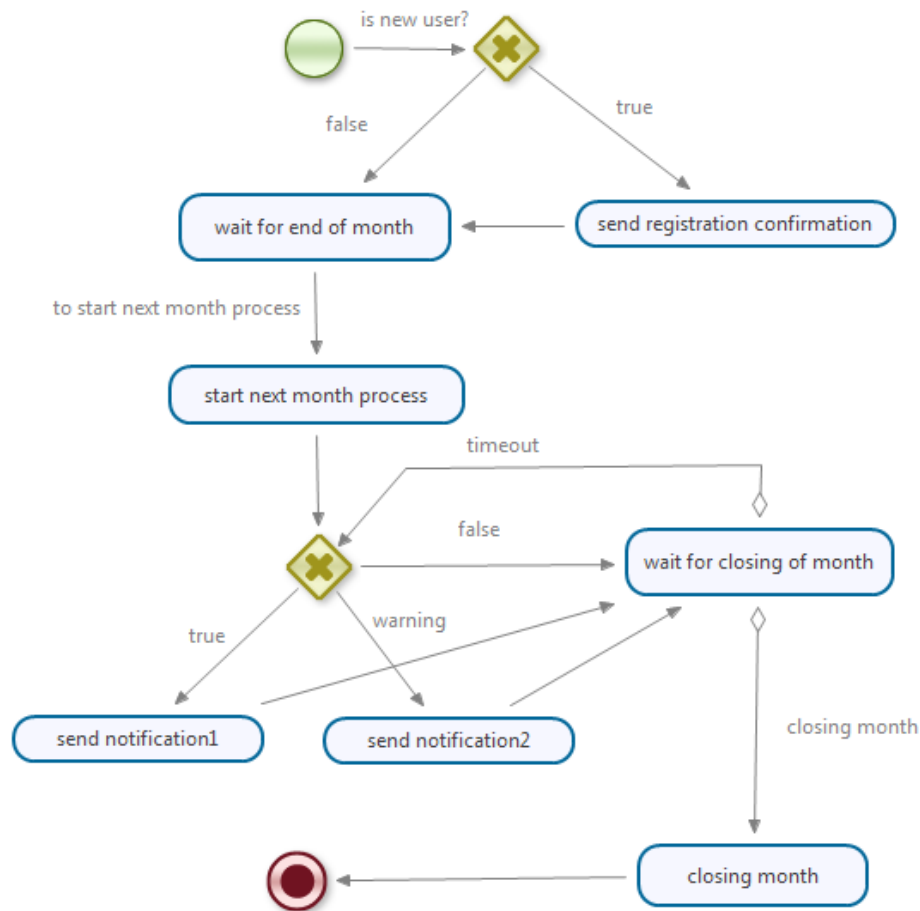


Abbildung 6.9.: Monatlicher Prozess

gegebenenfalls Warnungen an den Benutzer geschickt, die ihn auf eventuelle Fehler hinweisen, wie wenn er z.B. zu viele Stunden an einem Tag eingetragen hat oder am Ende des Monats seine vorgeschriebene Arbeitszeit überschreitet. Die hierfür benötigten Daten werden in der Serviceschicht berechnet, auf die im folgenden Abschnitt näher eingegangen wird.

### 6.2.2.3. Serviceschicht

Die Serviceschicht liefert der Prozessschicht alle nötigen Daten und Auskünfte, die diese in der weiteren Verwendung benötigt, wie schon im Ablauf des ersten Projekts erläutert. Hierzu gehören z.B. das Erzeugen von Warnungen, die unter bestimmten Umständen ausgegeben werden sollen, dem Berechnen von Ferientagen des Monats, Verteilung der restlichen Arbeitsstunden eines Monats auf das Default-Projekt, etc. Die Funktionalitäten der Serviceschicht wurden in folgende Packages aufgeteilt:

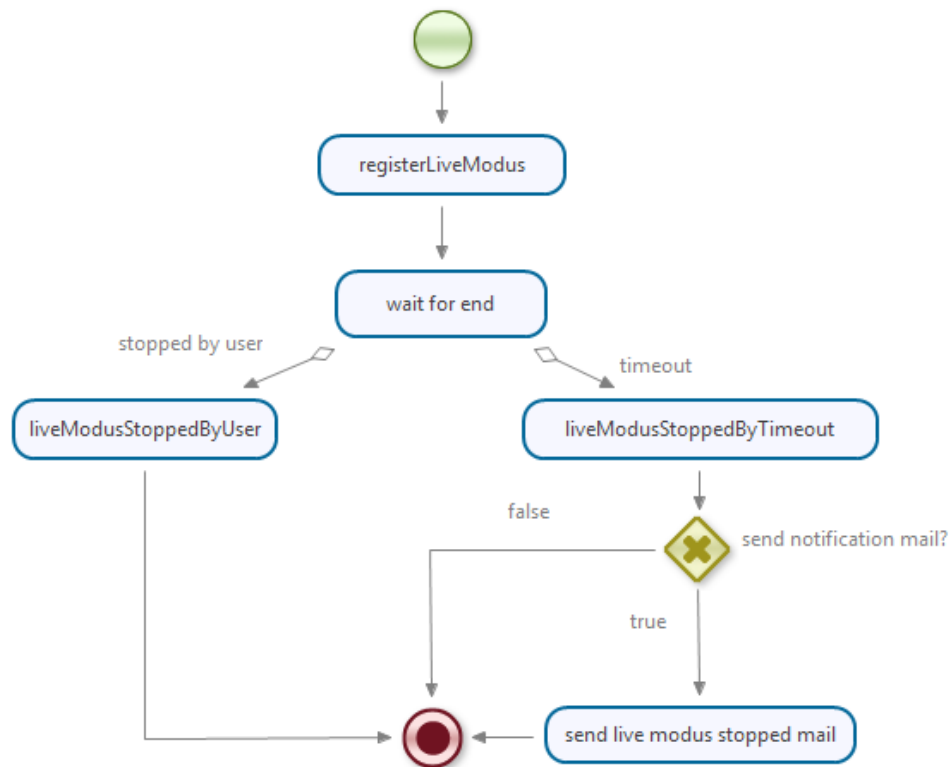


Abbildung 6.10.: Live-Modus Prozess

- constants: hier sind die benötigten Parameter für das Importieren bzw. Exportieren der Excel-Stundenzettel enthalten.
- export: beinhaltet die Funktionen für das Importieren bzw. Exportieren der Excel-Stundenzettel. Es gibt einen XLS-Controller, der das Schreiben bzw. Lesen eines Excelsheets ermöglicht.
- service: hier sind die Funktionen für das Speichern oder Lesen von Daten auf der Datenbank enthalten.
- util: stellt verschiedene oft verwendete Funktionen bereit. Es gibt unter anderem eine *DateUtil*-Klasse, eine *ServiceUtil*-Klasse und eine *WarningUtil*-Klasse. Während die *DateUtil*-Klasse sich um allgemeine Operationen bezüglich eines Datums kümmert, realisiert die *ServiceUtil*-Klasse allgemeine Funktionen für eine Datenbank. Die *WarningUtil*-Klasse stellt Funktionen rund um

die Benachrichtigungen bereit, wie z.B. das Versenden einer Benachrichtigung als E-Mail.

#### 6.2.2.4. Datenbankschicht

Das Datenmodell (siehe Abbildung 6.11) besteht aus:

- **Personen:** Die Stammdaten einer Person werden in *Person* aufgenommen. Zusätzlich gibt es zu jeder Person einen Benutzernamen und ein Passwort, welche von JAAS (s. [Oracle, a]) zur Authentifizierung verwendet werden.
- **Projekten:** Ein Projekt besteht in unserem Fall aus einer Projektkennzahl und einem Projektleiter bzw. -verantwortlichen, der am Ende des Monats die Arbeitszeiten gegenzeichnet. Projekte sind global gültig und werden keiner Person zugeordnet. Die Zuordnung zu einer Person geschieht über die Projektoptionen.
- **Projektoptionen:** Projektoptionen verbinden vorhandene (globale) Projekte und Benutzer miteinander. In einer Projektoption wird das Ende der Registrierungszeit, die zu leistende Arbeitszeit eines Benutzers für das Projekt und ein benutzerspezifischer Name für das Projekt gespeichert.
- **Workingsessions:** Eine Workingsession gibt die Arbeitszeit eines Benutzers für ein bestimmtes Projekt eines Tages an. Für den Fall, dass an einem Tag mehrfach Zeiten zu einem Projekt gebucht werden, werden die Zeiten in einer Workingsession aufaddiert, so dass es immer genau eine Workingsession je Tag und Projekt gibt.
- **Urlaubstage:** Ein Urlaubstag wird ausschließlich durch ein Datum repräsentiert, welches einem Benutzer zugeordnet ist.
- **Feiertage:** Ein Feiertag wird ausschließlich durch ein Datum repräsentiert, welches aus der hochgeladenen Excel-Datei ausgelesen wird und global in der Applikation gültig ist.
- **Warnungen:** Eine Warnung wird einem Benutzer zugeordnet und beinhaltet ein Datum, das betroffene Projekt und eine Nachricht. Ein zusätzilches Flag gibt an, ob die Warnung vom Benutzer als *gelesen* markiert wurde.
- **Rollen:** Eine Rolle besitzt lediglich einen Namen und eine Relation zwischen Benutzern und Rollen. Diese Zuordnung kann beispielsweise von JAAS zur Authorisierung verwendet werden.

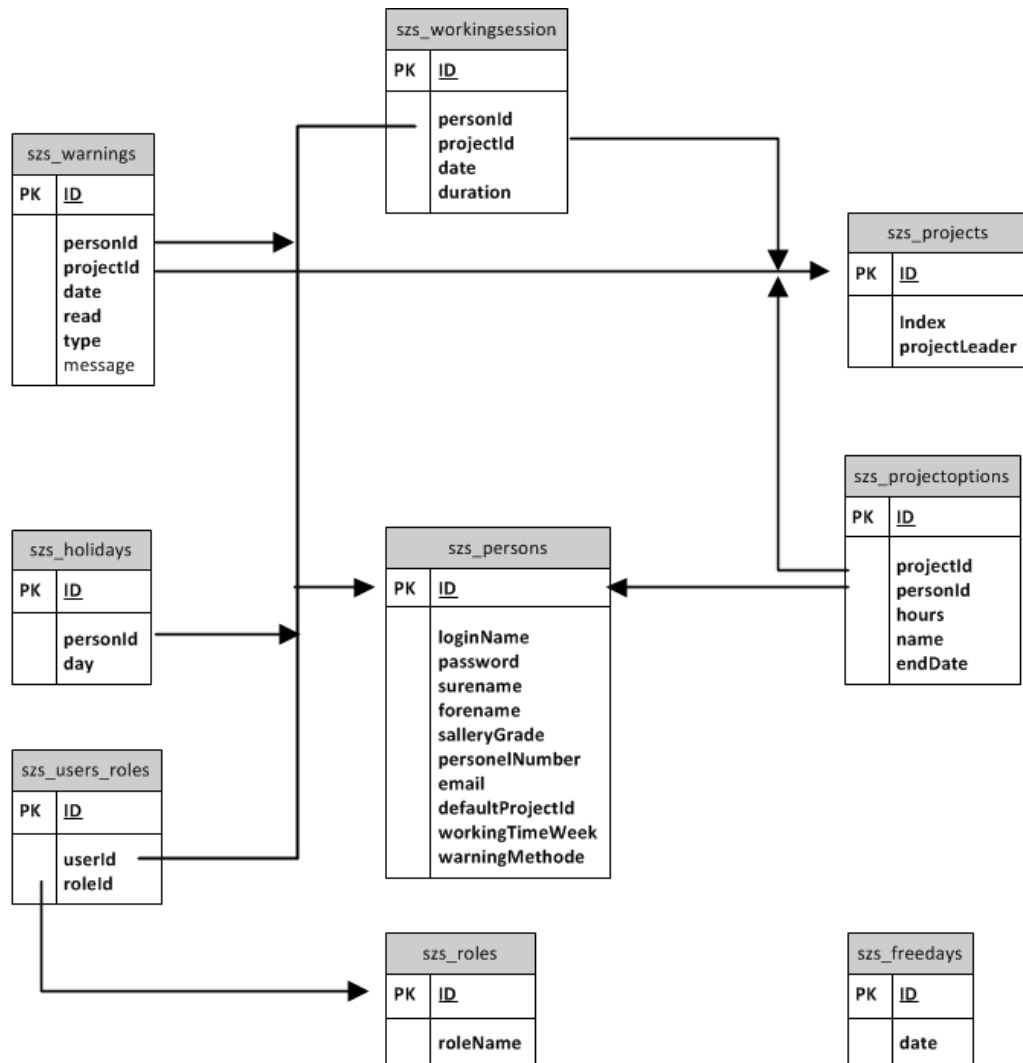


Abbildung 6.11.: Datenmodell Projekt 2

### 6.2.2.5. Webschicht

Nachdem die Implementierung abgeschlossen war, waren die folgenden aus den Tasks geforderten Oberflächen in der Webschicht entstanden, von denen einige ausgewählte hier durch einen Screenshot dargestellt werden sollen. Die GUI ermöglicht es dem Benutzer, verschiedene Funktionen auszuführen. Das Layout ist sehr übersichtlich gestaltet. Es hat hauptsächlich eine Menüleiste, eine linke Spalte für diverse Info-Anzeigen und eine Inhaltsspalte. Für jede Funktion steht ein Menüpunkt, der als Reiter dargestellt ist, bereit. Beim Starten der Anwendung muss sich der Benutzer einloggen. Dafür steht die Login-Seite. Falls er noch kein Konto besitzt, muss er sich registrieren. Von dort aus gelangt er auf die Login-Seite. Nachdem er sich erfolgreich eingeloggt hat, werden die folgenden Menüpunkte je nach Berechtigung eingeblendet:

- Neuen Benutzer registrieren



- Benutzer anmelden
- Neues Datenblatt hochladen: diese Funktionalität erlaubt es eine neue Excel-Vorlage für das angegebene Jahr hochzuladen. Diese Vorlage wird dann später verwendet um den Stundenzettel-Download bereitzustellen.
- Übersicht über die Arbeitszeiten (s. Abbildung 6.12): hier wird eine Übersicht über die Arbeitszeiten angezeigt. Weiterhin ist es möglich Stunden bzw. Urlaub einzutragen, zu löschen oder zu ersetzen. Das Herunterladen der Stundenzettel als Excel-Datei ist ebenfalls möglich. Die Warnungen (s. Abbildung 6.13) werden, falls vorhanden, ganz oben angezeigt. Außerdem kann der Benutzer von hier aus den Live-Modus starten, indem er in den tabellarischen Stundenzetteln auf den Projektnamen klickt.

**Arbeitszeitverwaltung**  
Ein Werkzeug zum Erfassen von Projektarbeitszeiten für die TU Dortmund

Übersicht | Einstellungen

Februar Freitag  
25

Personalnr.: 666  
Wochenarbeitszeit: 37.83  
0  
Download Excel

Februar 2011	Di 1	Mi 2	Do 3	Fr 4	Sa 5	So 6	Mo 7	Di 8	Mi 9	Do 10	Fr 11	Sa 12	So 13	Mo 14	Di 15	Mi 16	Do 17	Fr 18	Sa 19	So 20	Mo 21	Di 22	Mi 23	Do 24	Fr 25	Sa 26	So 27	Mo 28	Total
haushalt (12345)	8:00	5:00	3:00				U	U	U	U	U			U	U	U													21:00
stuze (2)							U	U	U	U	U			U	U	U													
Stundenzettel (4321)			3:00	6:00	8:00		U	U	U	U	U			U	U	U													17:00
Gesamtstunden pro Tag:	8:00	8:00	9:00	8:00																					5:00				38:00

Stunden eintragen  
Projekt:   
Datum: 25.02.2011  
Dauer:

Urlaub eintragen  
von:   
bis:

Eintragen | Wert ersetzen | Löschen | Eintragen | Entfernen

Copyright © 2000 - 2010 TU Dortmund / PG-551 / Impressum / Datenschutzerklärung  
Layout based on YAML

tu technische universität dortmund

Abbildung 6.12.: Übersicht

- Einstellungen (s. Abbildung 6.14): hier kann der Benutzer seine persönlichen Daten und Einstellungen für die Anwendungen (wie z.B. das Default-Projekt oder die Art der Benachrichtigungen bei Warnungen) ändern. Außerdem können hier neue Projekte hinzugefügt werden.
- Projekt aus Liste auswählen zur Übernahme in den persönlichen Stundenzettel



### 6.2.3. Zusammenfassung

Zusammenfassend kann man sagen, dass die Arbeitszeitverwaltung erfolgreich abgeschlossen wurde. Das Programm ist lauffähig und unterstützt alle gewünschten Funktionalitäten, die von höchster oder mittlerer Priorität waren. Bei den Anforderungen mit niedrigster Priorität wurde die Unterstützung eines Profils für Projektleiter zum Kenntlichmachen des Abschlusses eines Projekts für einen bestimmten Monat jedoch nicht umgesetzt.

Eine wichtige Entwicklung, die im Laufe dieses Projekts notwendig wurde, war die Ernennung eines Projektleiters, ohne den der Überblick über die noch zu verteilenden Aufgaben nicht gegeben gewesen wäre. Auch die Umstellung vom PivotalTracker auf Redmine war eine nicht zu vernachlässigende Entwicklung. Mit Hilfe von Redmine ist es der Projektgruppe nun möglich zu sehen, wer gerade an welcher Aufgabe arbeitet und darüberhinaus ist es den Teilnehmern - insbesondere also auch dem Projektleiter - möglich, anderen Teilnehmern eine Aufgabe zuzuweisen, was eine ungleiche Verteilung der Aufgaben vermeiden lässt. Weitere Informationen zu der Umstellung auf dieses Tool sind dem Kapitel 4 zu entnehmen.

## 6.3. Projekt 3: Getränkeverwaltung

Ziel des dritten Projektes war es, die bisher durch Excel-Tabellen realisierte Getränkeverwaltung unseres Kunden in eine prozessgesteuerte Webanwendung zu migrieren. Erneut wurde die Architektur des in Abschnitt 5.1 beschriebenen Schichtenmodells eingesetzt. Auf die Verwendung der einzelnen Schichten innerhalb dieses Projekts wird im Abschnitt 6.3.2 eingegangen.

### 6.3.1. Beschreibung

#### 6.3.1.1. Aufgabenstellung

Die Getränkeverwaltung des Kunden bestand bisher aus einer aushängenden Strichliste und einer zur Verwaltung und Abrechnung genutzten Excel-Tabelle, im Folgenden als Verwaltungsdokument bezeichnet. Die Aufgabe der Projektgruppe im dritten Projekt bestand nun darin, eine Webanwendung zu entwickeln, welche die Funktionalität des Verwaltungsdokuments implementiert und um diverse Anforderungen erweitert. Die Strichliste sollte in ähnlicher Form erhalten bleiben und von der Anwendung automatisch generiert werden. Die an die Webanwendung gestellten Anforderungen werden in Abschnitt 6.3.1.3 weiter beschrieben.

#### 6.3.1.2. Beschreibung der Excel-Datei

**Die Strichliste** Teil der bisherigen Getränkeverwaltung unseres Kunden war eine Strichliste (siehe Abbildung 6.15). Diese wurde ausgedruckt und genutzt, um die Menge an Getränken festzuhalten, die durch bekannte Kunden konsumiert wurden.

Die so erhobenen Daten wurden in unregelmäßigen Abständen in das im Folgenden näher beschriebene Verwaltungsdokument übertragen.

**Preise**

20.08.2011

Cola/Fanta/Sprite Light/Zero	1.0L	1,00 €
Wasser (Schloss Quelle)	1.0L	0,50 €
Apfelschorle	0.75L	0,80 €
Bier	0.5L	0,80 €

Einlage: 5 € - Schlüssel: im R130 - Ansprechpartner für den Getränkediens: Alexander Armut (R130)

Name	Cola/ Fanta/Sprite N/Z/L	Wasser Schloss Quelle	Apfelschorle	Bier
Armut, A.				
Blöd, B.				
Clever, C.				
Dämlich, D.				
Ehrlich, E.				
Fröhlich, F.				
Gärtner, G.				
Hufschmidt, H.				
Irsinn, I.				
Jodler, J.				
Kolumna, K.				
Lügner, L.				
Mustermann, M.				
Gäste				

\*\*\*\*\* Bitte unbedingt Pfandflaschen zurückgeben! \*\*\*\*\*

Abbildung 6.15.: Original Strichliste

**Das Verwaltungsdokument** Das Verwaltungsdokument enthält zwei Tabellen, die für unterschiedliche Teilaspekte der Verwaltung zuständig sind. Die Tabelle „Abrechnung“ (s. Abb. 6.16) wird genutzt, um das eingezahlte Guthaben der registrierten Kunden sowie den Kassenstand, Warenwert und Pfandwert festzuhalten. Der Stand der Barkasse (oben mitte), die Verkaufspreise (oben rechts), die Spalte „Konto Alt“ (Tabelle in der Mitte) sowie die Warenbestände (unten) enthalten konkrete Werte, müssen also manuell aktuell gehalten werden. Sind diese Bestandsdaten korrekt eingetragen, so werden die daraus abgeleiteten Werte (Warenwert, etc.) automatisch berechnet. Um eine Strichliste abzurechnen, müssten dann nur noch die Verbrauchsmengen in die entsprechenden Getränkespalten übertragen werden, sowie die Lagerbestände angepasst werden, die neuen Kontostände werden dann automatisch berechnet. Um die Tabelle auf die nächste Iteration der Abrechnung vorzubereiten müssen die neu berechneten Kontostände noch manuell in die Spalte „Konto Alt“ übertragen werden und anschließend die Verbrauchsangaben wieder aus den Getränkespalten entfernt werden.

Die Tabelle „Einkaufsaufträge“ (s. Abb. 6.17) wird genutzt, um Einkäufe zu planen und zu protokollieren. Beutzt wird sie, indem der Abschnitt des letzten getä-

<b>Stand:</b>	17.12.2010				
<b>zuletzt bearbeitet:</b>	17.12.2010				
<b>Beiträge (im Vor.)</b>	149,67 €	<b>Warenwert</b>	130,08 €	<b>Preisliste:</b>	
<b># Personen</b>	13	<b>Kasse (bar)</b>	124,78 €	Cola	1,00
<b>Einlage</b>	5,00 €	<b>Haben</b>	254,86 €	Wasser SQ	0,50
<b>Haben (Einlagen)</b>	65,00 €	<b>Saldo</b>	40,19 €	Bier	0,80
<b>Haben (soll)</b>	214,67 €			Apfelschorle	0,80
				Saft	1,30

Gesamtverbrauch									
Name	Cola	Wasser SQ	Bier	AS	Saft	Konto Alt	Konto Neu	Bezahlt	zuletzt am
Armut, Alexander						29,16 €	29,16 €		10.03.11
Blöd, Bernd						0,00 €	0,00 €		07.04.11
Clever, Clemens						15,65 €	15,65 €		17.12.10
Dämlich, Doris						0,64 €	0,64 €		17.02.11
Ehrlich, Elvira						2,00 €	2,00 €		17.02.11
Fröhlich, Felix						0,00 €	0,00 €		17.02.11
Gärtner, Georg						26,41 €	26,41 €		18.02.11
Hufschmidt, Hans						0,00 €	0,00 €		20.12.10
Irsinn, Ida						-0,80 €	-0,80 €		
Jodler, Johan						9,00 €	9,00 €		18.02.11
Kolumna, Karla						4,80 €	4,80 €		17.02.11
Lügner, Leander						19,58 €	19,58 €		13.10.10
Mustermann, Max						43,23 €	43,23 €		05.04.11
						0,00 €	0,00 €		
						0,00 €	0,00 €		
						0,00 €	0,00 €		

Warenwert	Pfand [#K]	Bier	Schloss Quelle	Steinsieker	Cola	Saft
Pfand gesamt:	0	0	6	4	4	0
Inhalt gesamt:	46,20	0,00	0,00	19,80	13,20	13,20
Gesamtsumme:	83,88	0,00	0,00	27,00	14,88	42,00
	130,08					

Sonstiges: 01.04.08 Sackkarre für 20 Euro gekauft, ist als Alexander Armuts Guthaben verrechnet  
 Geldkassette für 8 Euro gekauft, ist als Alexander Armuts Guthaben verrechnet  
 28.05.08 Nina Neuer hat 16 Euro Einlage und 1,90 Euro Guthaben erhalten.  
 29.08.08 Olga Obermann's Einlage wurde zum Ausgleich des Guthabens verrechnet.

Abbildung 6.16.: Verwaltungsdokument - Tabelle „Abrechnung“

tigten Einkaufes am Ende der Tabelle dupliziert und angepasst wird. Jede Zeile behandelt je eine Getränkesorte. Die Spalten „Pfand #[K]“ und „Kauf #[K]“ enthalten die Anzahlen der abzugebenden Leerkästen bzw. neu zu kaufenden Kästen, während die restlichen Spalten weitere Informationen, wie Name, Pfandwert und erwarteter Einkaufspreis zu der jeweiligen Getränkesorte, enthalten. Würde ein Einkauf getätigt, müssen die Änderungen an Lager- und Kassenbeständen manuell in die Abrechnungstabelle übertragen werden. Falls sich die Einkaufspreise geändert haben, sollten diese in der Einkaufstabelle angepasst werden, bevor der Einkauf als erledigt markiert und mit einem Datum versehen wird. Die Tabelle wird fortlaufend nach unten erweitert und enthält dementsprechend eine Historie über alle getätigten Einkäufe.

Die in beiden Dokumenten vorhandenen Datensätze, wie die angebotenen Getränke und ihre Verkaufspreise, sowie die Namen der bekannten Benutzer, müssen manuell konsistent gehalten werden.



### 6.3.1.3. Anforderungsanalyse

Die Anforderungsanalyse wurde, wie bereits bei den vorherigen Projekten, während einer persönlichen Befragung der Kunden durchgeführt. Daraus ergaben sich folgende Anforderungen:

#### 1. Strichliste

Die auszudruckende, physisch aushängende Strichliste soll weiterhin integraler Bestandteil des Gesamtsystems bleiben. Mit dem Austauschen der Strichliste soll ein neuer *Abrechnungszeitraum* beginnen. Diese Bindung hat grossen Einfluß auf die Modellierung der Prozessschicht, mehr dazu in Abschnitt 6.3.2.3.

#### 2. Online-Strichliste

Die Anwendung soll Benutzern die Möglichkeit geben, über ein Webinterface „Striche zu machen“ und ihre online getätigten Getränkäufe einzusehen.

#### 3. Getränkmodellierung

Die Getränke sollen so modelliert werden, dass unterschiedliche, im Einzelhandel erhältliche, *Getränkessorten* als zum Weiterverkauf äquivalente *Getränkekategorien* zusammengefasst werden können. Beispielsweise könnten „Coca Cola 1,0L“ und „Pepsi Cola 1,0L“ als „Cola“ weiterverkauft werden.

#### 4. Benutzerverwaltung

Die Anwendung soll Benutzer mit unterschiedlichen Berechtigungen unterstützen. *Verwalter* haben Zugriff auf alle Funktionen der Anwendung und dürfen *normale Benutzer* registrieren. *Normale Benutzer* dürfen Getränke kaufen und haben Zugriff auf ihren eigenen Kontostand sowie eine Historie ihrer getätigten Transaktionen. *Einkäufer* sind spezielle Benutzer denen Einkaufsaufträge zugeordnet werden können.

#### 5. Kontoverwaltung

Sowohl der Stand der Barkasse, als auch die Kontostände der Benutzer sollen durch die Anwendung verwaltet werden. Es soll einsehbar sein, wann ein Benutzer zuletzt eingezahlt hat. Außerdem soll die Anwendung in der Lage sein Rechnungen an Benutzer zu verschicken und getätigte Einzahlungen vorhandenen Rechnungen zuzuordnen. Ein- und Auszahlungen, die keiner Rechnung zugeordnet werden, sollen auch möglich sein.

#### 6. Lagerverwaltung

Die Anwendung soll den Bestand an gelagerten Getränken und Pfand selbstständig verwalten. Es soll außerdem möglich sein, den Lagerbestand bei Bedarf durch eine Inventur manuell anzupassen. Eine Warnung bei niedrigem Lagerbestand wurde ebenfalls gewünscht.

#### 7. Einkaufsabwicklung

Über die Anwendung soll es möglich sein, Einkaufsaufträge zu erstellen und

einem Einkäufer zuzuweisen. Die zu erwartenden Kosten sollen dabei aus den Einkaufspreisen, die den gewählten Getränkesorten zugeordnet sind, und ihren Pfandwerten berechnet werden. Einkäufe können - beispielsweise aufgrund von Preisänderungen, mangelndem Angebot oder ähnlichem - abweichend von dem zugehörigen Einkaufsauftrag oder ohne Einkaufsauftrag abgerechnet werden. In der Regel werden Einkäufe in der Einheit *Kästen* beauftragt und abgerechnet. Ein Wechsel auf *Flaschen* soll für Spezialfälle möglich sein. Einkäufer sollen pro gekauftem Kasten eine Bonuszahlung von zur Zeit einem Euro gutgeschrieben bekommen.

#### 8. **Versand von E-Mails**

Um neu registrierten Benutzern ihre Zugangsdaten mitzuteilen sowie Rechnungen und Einkaufsaufträge zu verschicken, soll die Anwendung fähig sein E-Mails an Benutzer zu versenden.

#### 9. **Graphische Aufbereitung von Statistiken**

Viele Daten und ihre Änderungen sollen festgehalten werden, um aus ihnen Statistiken - beispielsweise über Verbrauch oder Einzahlungen - zu berechnen. Sofern dies sinnvoll möglich ist, sollen diese Statistiken graphisch aufbereitet werden.

10. **Webservices** Gegen Ende der Implementationsphase wurde von den Kunden die zu Beginn bereits angedeutete Anforderung konkretisiert, die von der Webschicht genutzten Funktionen aus Service- und Prozessschicht auch als Webservices anzubieten. Diese Problematik und ihre Lösung wird in Abschnitt 6.3.2.5 näher beschrieben.

### **6.3.2. Ablauf**

#### **6.3.2.1. User Stories**

Wie im vorherigen Projekt wurde der PG das neue Excelsheet vom Kunden vorgestellt. Um die Wünsche des Kunden umzusetzen, wurde eine Anforderungsanalyse gemacht, woraus dann die User Stories entstanden sind. Diese wurden dann vom Kunden priorisiert. Im weiteren Verlauf folgen die User Stories nach Prioritäten aufgelistet.

#### **User Stories mit Priorität 1**

1. **Anlegen von Kundenkonten:** Der Verwalter soll über ein Formular neue Benutzer anlegen können. Außerdem soll er die Möglichkeit besitzen, vorhandene Benutzer zu deaktivieren.
2. **Ausdrucken einer leeren Strichliste:** Der Verwalter soll eine leere Strichliste ausdrucken können, die die Namen aller aktiven Benutzer enthält sowie



alle Getränkekategorien, die im System angelegt worden sind und im Lager vorhanden sein sollten.

3. **Einsicht in Kundenkonten (aktueller Stand, Historie):** Der Verwalter soll Einblick in die Konten der Benutzer haben, um sich den aktuellen Kontostand und die Zahlungshistorie anschauen zu können.
4. **Geld auf Kundenkonto einzahlen (bar):** Der Verwalter soll jederzeit die Möglichkeit haben, über ein Formular einen bestimmten Betrag auf das Konto eines Benutzers als Einzahlung verbuchen zu können.
5. **Geld auf Kundenkonto einzahlen (Rechnung):** Der Verwalter soll Einzahlungen für einen Benutzer vornehmen können, für die er dem Benutzer vorher eine Rechnung geschickt hat.
6. **Rechnungsmails erzeugen und verschicken:** Der Verwalter soll zu beliebigen Zeitpunkten Benutzern Rechnungen per E-Mail schicken können.
7. **Rundmails versenden:** Der Verwalter soll zu beliebigen Zeitpunkten Benutzern und Verwaltern E-Mails versenden können.
8. **Rechnungshistorie einsehen:** Der Verwalter soll sich jederzeit eine Rechnungshistorie anzeigen lassen können, in der Rechnungen als verschickt, noch nicht bezahlt oder bereits bezahlt markiert sind.
9. **Strichliste in Anwendung übertragen:** Der Verwalter soll die Striche der physikalischen Strichliste in die online geführte Strichliste übertragen können. Dies geschieht nur zum Ende eines Abrechnungszeitraums.
10. **Einkaufsauftrag erzeugen und Einkäufer bestimmen:** Der Verwalter soll jederzeit einen Einkaufsauftrag erzeugen und diesen einem Einkäufer seiner Wahl zuweisen können.
11. **Getätigten Einkauf in Anwendung übertragen:** Der Verwalter soll nach einem Einkauf das Lager entsprechend dem Einkauf aktualisieren können. Außerdem soll der Einkauf mit der Kasse verrechnet werden.

12. **Bonuszahlung an Einkäufer tätigen:** Nach erfolgtem Einkauf soll der Verwalter eine Bonuszahlung an den Verkäufer vornehmen können.
13. **Getränkessorten anlegen:** Der Verwalter soll neue Getränkessorten anlegen können. Dazu zählen sowohl Getränkekategorien als auch konkrete Getränkessorten.
14. **Einkäuferstatus für Benutzer vergeben:** Der Verwalter soll beliebigen Benutzern die Rolle des Einkäufers vergeben können.
15. **Verkaufspreis für Getränke festlegen:** Für jede Getränkekategorie soll ein Verkaufspreis festgelegt werden können. Verkaufspreise können auch innerhalb eines Abrechnungszeitraums geändert werden. Die Änderungen greifen dann allerdings erst zum Start des nächsten Abrechnungszeitraums.
16. **Standardeinkaufspreis für Getränke festlegen:** Für jede Getränkessorte soll ein Standardeinkaufspreis festgelegt werden können.

## **User Stories mit Priorität 2**

1. **Ein- und Auszahlungen auf Konten / Kasse mit Notiz vermerken:** Jede durch den Verwalter durchgeführte Konten- oder Kassenbewegung soll vom Verwalter mit einer Notiz versehen werden können.
2. **Lagerbestand online einsehen:** Der Verwalter soll jederzeit den aktuellen Lagerbestand online einsehen können. Die Aktualität bezieht sich dabei nur auf die online getätigten Getränkekäufe.
3. **Lagerbestand nach Onlineeinkauf aktualisieren:** Nachdem ein Benutzer online ein Getränk gekauft hat, soll dieses vom Lagerbestand abgezogen werden.
4. **Lagerbestand bei Bedarf manuell anpassen:** Der Verwalter soll jederzeit die Möglichkeit haben, den aktuellen Lagerbestand manuell anzupassen z.B. im Falle einer Inventur.

5. **Getränke für Gäste / anonyme Benutzer eintragen:** Der Verwalter soll die Möglichkeit haben, Getränke auf ein anonymes Gästekonto zu buchen.
6. **Untere Grenze für Kundenkonten anlegen:** Der Verwalter soll eine untere Grenze für Kundenkonten festlegen können, ab welcher er über den Kontostand eines Benutzers informiert wird.
7. **Online Strichliste führen:** Kunden sollen die Möglichkeit haben, ihre Getränke auch in einer Online-Strichliste vermerken zu können. Dabei sieht jeder Benutzer allerdings nur die von ihm konsumierten Getränke.
8. **Erhalt von automatischer Warnung bei niedrigem Lagerstand:** Der Verwalter soll automatisch gewarnt werden, wenn Getränke einer Kategorie knapp werden.
9. **Einkauferrangliste einsehen:** Der Verwalter soll die Möglichkeit haben, eine Einkauferrangliste angezeigt zu bekommen, um zu sehen, welche Einkäufer wie oft und wieviel eingekauft haben.
10. **Benutzern Verwaltungsrechte geben:** Der Verwalter soll andere Benutzer ebenfalls zu Verwaltern ernennen können.
11. **Ein- und Auszahlungen in die Kasse vornehmen:** Der Verwalter soll jederzeit die Möglichkeit haben, beliebige Beträge in die Kasse ein- oder aus der Kasse auszahlen zu können. Dazu zählen z.B. Trinkgeld oder außerplanmäßige Anschaffungen.
12. **Kassenstand einsehen:** Der Verwalter soll jederzeit den aktuellen Kassenstand einsehen können. Dazu gehört auch der aktuelle Warenwert und der Wert, der in Form von Pfand vorhanden ist.

### **User Stories mit Priorität 3**

1. **Getränke-Toplisten einsehen:** Jeder Benutzer soll Ranglisten der am meisten konsumierten Getränke abrufen können.

2. **Verbrauchsstatistik pro Getränk einsehen:** Jeder Benutzer soll für einen beliebigen Zeitraum Verbrauchsstatistiken zu den konsumierten Getränken abrufen können.
3. **Kontohistorie einsehen:** Jeder Benutzer soll die eigene Kontohistorie einsehen können.
4. **Kontostand ansehen:** Jeder Benutzer soll seinen aktuellen Kontostand einsehen können.
5. **Durchschnittsverbrauch pro Woche/Monat einsehen:** Der Verwalter soll die Möglichkeit haben, den durchschnittlichen Getränkeverbrauch pro Woche/Monat einsehen zu können.
6. **Monatlicher Getränkekonsum pro Kunden einsehen:** Der Verwalter soll den monatlichen Getränkekonsum pro Benutzer einsehen können.
7. **Durchschnittseinzahlungen der Kunden einsehen:** Der Verwalter soll pro Kunden sehen können, wieviel dieser durchschnittlich einzahlt.
8. **Verbrauchshochrechnung für einen bestimmten Zeitraum:** Der Verwalter soll sich eine Verbrauchshochrechnung für einen bestimmten Zeitraum anzeigen lassen können. Die Verbrauchshochrechnung liefert Informationen darüber, wie lange der aktuelle Warenbestand schätzungsweise reichen wird bzw. wann wieder nachgekauft werden muss.

#### 6.3.2.2. Datenbankschicht

Das Datenmodell (siehe Abbildung 6.18) besteht aus:

- **Abrechnungszeitraum (Accounting Period):** Ein Abrechnungszeitraum wird durch ein Anfangs- und durch ein Enddatum bestimmt. Der aktuelle Abrechnungszeitraum hat ein leeres Enddatum.
- **Getränkekategorie (Drink Category):** Eine Getränkekategorie repräsentiert die gleiche Art eines Getränks, welche jedoch verschiedene Marken haben kann (z.B. sind Cola und Cola Light zwei verschiedene Getränkekategorien; Coca Cola und Pepsi Cola gehören dagegen zur Kategorie Cola). Solch eine Kategorie besteht aus einem Namen, einem Verkaufspreis, einer minimalen

Mengenangabe und einem aktiv-Flag. Durch dieses kann man eine Getränke-  
 kategorie als inaktiv kennzeichnen, wenn diese Kategorie nicht mehr angeboten  
 wird. Zusätzlich wird eine Getränkesorte in dieser Kategorie als Standardsorte  
 gespeichert.

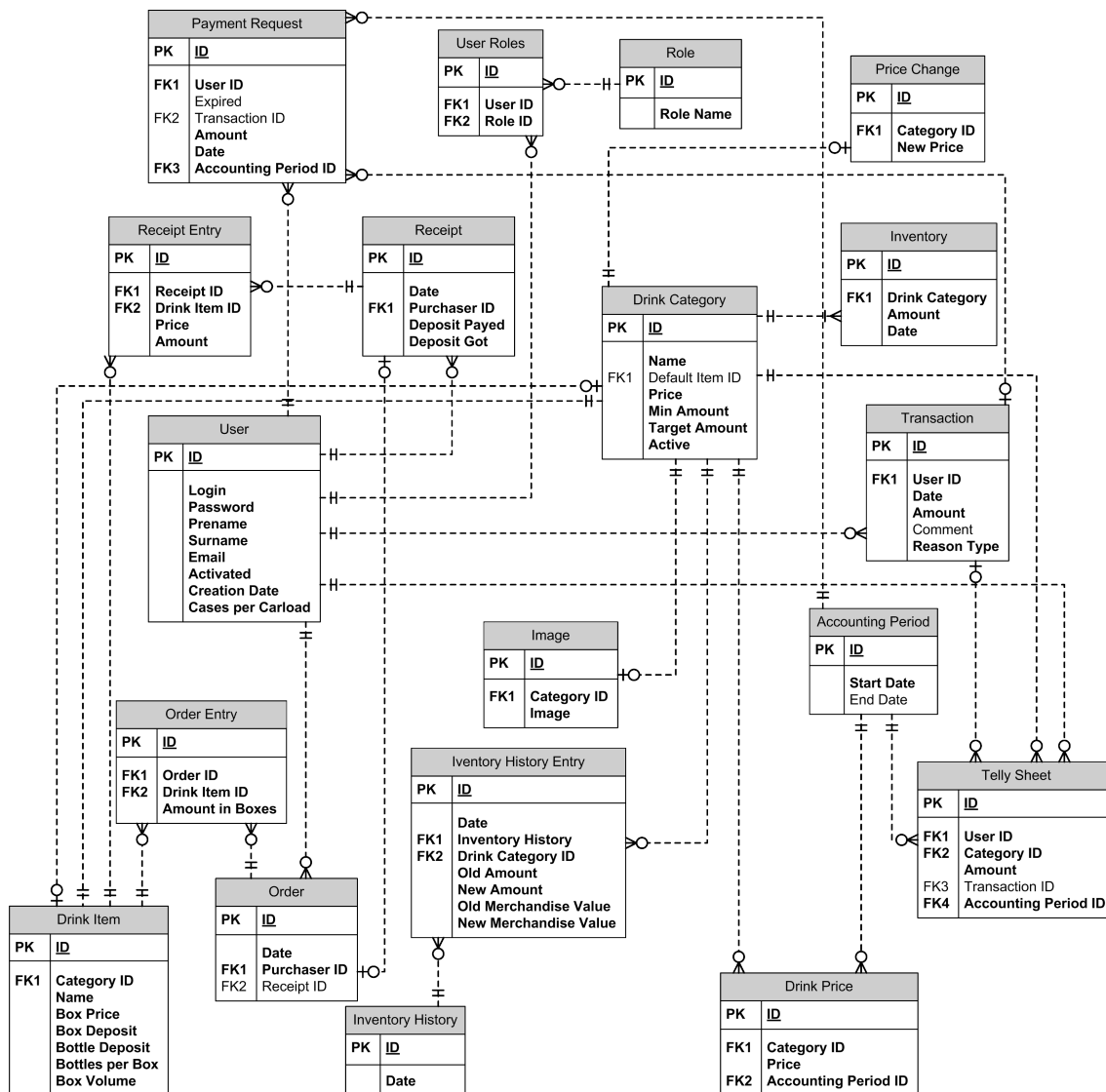


Abbildung 6.18.: Datenmodell Projekt 3

- **Getränkessorte (Drink Item)**: Eine Getränkesorte repräsentiert die gleichen Getränke, die eine verschiedene Marke haben (z.B. Coca Cola, Pepsi Cola). Getränkesorten werden einer Getränkekategorie zugeordnet. Die für den Einkauf relevanten Informationen (Einkaufspreis, Pfand etc.) werden auch mit in die Getränkesorte aufgenommen.
- **Verkaufspreis (Drink Price)**: Diese Tabelle enthält die Verkaufspreise al-

ler Getränkekategorien aus den früheren Abrechnungszeiträumen. Nach einer Preisänderung bleiben die alten Preise in der Datenbank, damit sie für Statistiken zur Verfügung stehen.

- **Bild (Image):** Jede Getränkekategorie besitzt ein Bild, welches im Getränkeautomaten angezeigt wird. Das Bild einer Getränkekategorie wird als Binary Large Object (BLOB) einer png-Datei in der Datenbank gespeichert.
- **Lagerbestand (Inventory):** Der Lagerbestand gilt für eine Getränkekategorie zu einem bestimmten Datum. Es gibt für jede Getränkekategorie nur einen aktuellen Eintrag des Lagerbestands. Die restlichen Einträge sind für die Statistik.
- **Inventur (Inventory History):** Eine Inventur besteht aus mehreren Inventureinträgen und dem Datum, an dem die Inventur durchgeführt wurde.
- **Inventureintrag (Inventory History Entry):** Ein Inventureintrag nimmt den Lagerbestand und den Warenwert von einer Getränkekategorie vor, und nach der Inventur auf.
- **Einkaufsauftrag (Order):** Ein Einkaufsauftrag besteht aus mehreren Positionen und wird einem Einkäufer zugeordnet. Nachdem der Einkauf getätigt wurde, wird der Kassenzettel dem entsprechenden Einkaufsauftrag zugeordnet.
- **Einkaufsauftragseintrag (Order Entry):** In einem Einkaufsauftragseintrag wird festgelegt, wie viele Kästen einer Getränkesorte eingekauft werden sollen.
- **Zahlungsaufforderung (Payment Request):** Eine Zahlungsaufforderung umfasst den jeweiligen Benutzer, den zu zahlenden Betrag, das Datum und den Abrechnungszeitraum. Ein zusätzlicher Eintrag gibt an, ob die Aufforderung aktuell ist. Eine Zahlungsaufforderung ist nicht mehr aktuell, wenn sie noch nicht beglichen wurde, aber dem entsprechenden Benutzer schon eine neue Zahlungsaufforderung zugestellt wurde. Im System wird dann zunächst nur noch die aktuellere Zahlungsforderung angezeigt. Liegt bei einer Einzahlung eine offene Zahlungsaufforderung für den entsprechenden Benutzer vor, so wird die entsprechende Buchung dieser Zahlungsaufforderung zugeordnet.
- **Preisänderung (Price Change):** Der Verkaufspreis einer Getränkekategorie kann nur zum Anfang eines neuen Abrechnungszeitraums geändert werden. Bevor der alte Abrechnungszeitraum beendet wird, werden die neuen Preise zwar in den Preisänderungen gespeichert, sind aber im System noch nicht gültig. Sie werden übernommen sobald der Abrechnungszeitraum endet.

- **Kassenzettel (Receipt):** In einem Kassenzettel wird das Einkaufsdatum, der Einkäufer, der abgegebene Pfandwert bzw. der bezahlte Pfandwert gespeichert.
- **Kassenzetteleintrag (Receipt Entry):** Ein Kassenzetteleintrag besteht aus dem zugeordneten Kassenzettel, der betroffenen Getränkesorte, dem Einkaufspreis der Getränkesorte und der eingekauften Menge.
- **Rolle (Role):** Eine Rolle besitzt einen Namen und wird mehreren Benutzern zugeordnet. Diese Zuordnung geschieht über die Benutzer-Rolle-Tabelle.
- **Strichliste (Tally Sheet):** Eine Strichliste repräsentiert den Getränkeverkauf an einen Kunden. Ein Eintrag der Strichliste besteht aus der gekauften Getränkekategorie, der gekauften Menge und dem Abrechnungszeitraum. Eine Strichliste bleibt nur bis zum Ende des aktuellen Abrechnungszeitraums gültig. Nach dem Abrechnungszeitraum wird die alte Strichliste dem Benutzer nicht mehr angezeigt. Die alte Strichliste bleibt dann in der Datenbank für statistische Zwecke erhalten.
- **Buchung (Transaction):** Eine Buchung beschreibt eine Ein- oder Auszahlung eines Benutzers. Zu einer Buchung gehört ein Benutzer, das Datum, der Betrag, der Zahlungstyp und ein optionaler Kommentar. Buchungen mit der zugeordneten Benutzer-Id -1 werden als Buchungen der Barkasse interpretiert, und Änderungen des Pfandwertes werden durch Buchungen für die Benutzer-Id -2 realisiert.
- **Benutzer (User):** Die Stammdaten eines Benutzers werden in der Tabelle Benutzer aufgenommen. Zusätzlich gibt es zu jedem Benutzer einen Benutzernamen, und ein mit MD5 gehashtes Passwort, welches zur Authentifizierung verwendet wird. Ein Benutzer kann nicht gelöscht, sondern nur deaktiviert werden.
- **Benutzer-Rolle (User Roles):** In dieser Tabelle werden zu jedem Benutzer seine zugeordneten Rollen gespeichert. Ein Benutzer kann auch mehrere Rollen gleichzeitig haben. Es gibt die Rollen *Kunde* (Standardbenutzer), *Einkäufer* und *Verwalter* (Administrator). Alle Verwalter und Einkäufer sind gleichzeitig auch Kunden.

### 6.3.2.3. Prozessschicht

Aufgabe der Prozessschicht ist es, möglichst viel der eigentlichen Geschäftslogik der Anwendung zu kapseln, so dass möglichst wenig Anwendungslogik in der Tapestry- und Serviceschicht steckt. Dies ist allerdings nur bis zu einem gewissen Rahmen sinnvoll, da viele Anwendungsfälle nur eine geringe Komplexität aufweisen, so dass es in diesen Fällen wesentlich weniger Aufwand bedeutet, die Tapestryschicht direkt mit der Serviceschicht kommunizieren zu lassen. Die Vorteile der Prozessschicht, wie

z.B. einfaches Austauschen bzw. Umschreiben der Prozesse, können in diesen Fällen nicht ausgespielt werden, weshalb es durchaus legitim ist, hierbei auf die Nutzung der Prozessschicht zu verzichten.

Somit bestand eine der Hauptaufgaben zu Beginn des dritten Projekts darin, die User Stories in zwei Teile zu gliedern: In User Stories, deren Implementierung ohne Prozessschicht auskommt, und in User Stories, für die es sinnvoll und notwendig ist, jeweils einen eigenen Prozess zu definieren. Dabei wurden in einer ersten Diskussion folgende Prozesse identifiziert:

1. **Abrechnung** Dieser Prozess drückt zu Beginn eine neue, leere Strichliste aus und startet damit einen neuen Abrechnungszeitraum. Erst danach wird der alte Abrechnungszeitraum beendet, die alte Strichliste digitalisiert und die konsumierten Getränke verrechnet. Anschliessend sollen automatisiert Rechnungen an die Benutzer verschickt werden (siehe Abbildung 6.19).
2. **Auftragsverteilung** Dieser Prozess prüft den Warenbestand und schlägt darauf basierend einen Einkaufsvorschlag vor. Nach Anpassung des Einkaufsvorschlags soll der Prozess in Frage kommende Einkäufer vorschlagen, so dass der Verwalter anschliessend einen Einkäufer wählen und ihn mit dem Einkauf beauftragen kann (siehe Abbildung 6.20).
3. **Einkauf** Der Einkaufsprozess wird nach einem getätigten Einkauf gestartet und speichert den Einkauf in der Anwendung, indem die Ware ins Lager eingelagert, der Kassenstand angepaßt und eine Bonuszahlung an den Benutzer getätigt wird (siehe Abbildung 6.21).
4. **Konsumieren** Dieser Prozess bildet den Konsum eines Getränks durch einen Anwender ab. Nachdem der Anwender den Konsum eines Getränks in die Anwendung eingetragen hat, soll der Prozess das Konto des Anwenders belasten, den Lagerbestand aktualisieren und anschliessend eine Kontoprüfung für den Anwender durchführen (siehe Abbildung 6.22).
5. **Kontoprüfung** Im Rahmen des Kontoprüfungsprozesses soll bei Unterschreitung eines bestimmten Kontostandes automatisiert eine Rechnung an den Anwender geschickt werden (siehe Abbildung 6.23).

Diese Prozesse sind allerdings noch nicht als jBPM-Prozesse zu verstehen, sondern sollten uns zunächst nur dabei helfen, mögliche Prozesse zu identifizieren und auszuarbeiten, die später in jBPM-Prozesse überführt werden konnten. So wurden diese Prozesse in nachfolgenden Diskussionen teilweise ummodelliert, gestrichen oder durch andere Prozesse ersetzt.

Der überarbeitete Entwurf des Konsumieren-Prozesses, der in „Online purchase“ umbenannt wurde, sieht keinerlei Kontobelastungen mehr vor (siehe Abbildung 6.24). Somit entfällt innerhalb dieses Prozesses auch die Kontoprüfung. Stattdessen wird das konsumierte Getränk nun nur noch in der Online-Strichliste vermerkt



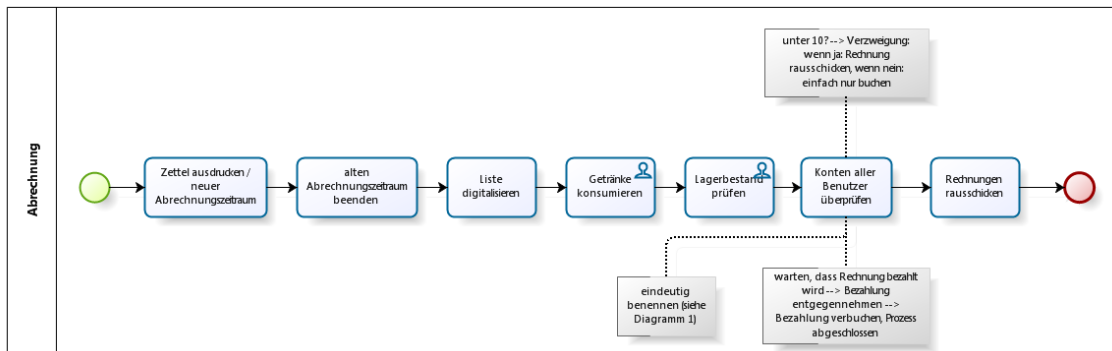


Abbildung 6.19.: Erster Entwurf: Abrechnungsprozess

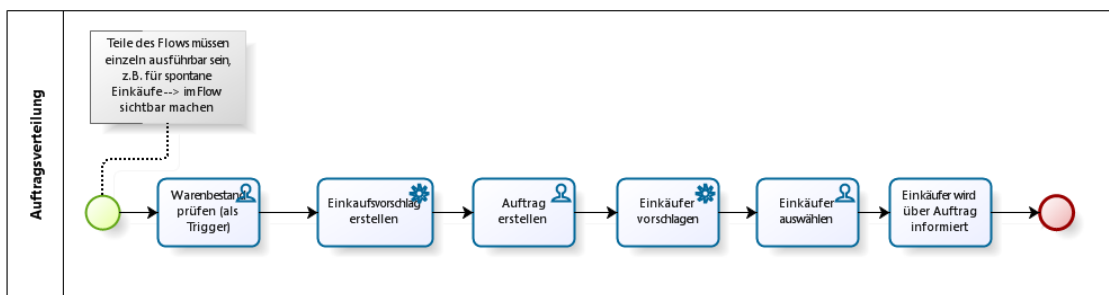


Abbildung 6.20.: Erster Entwurf: Auftragsverteilungsprozess

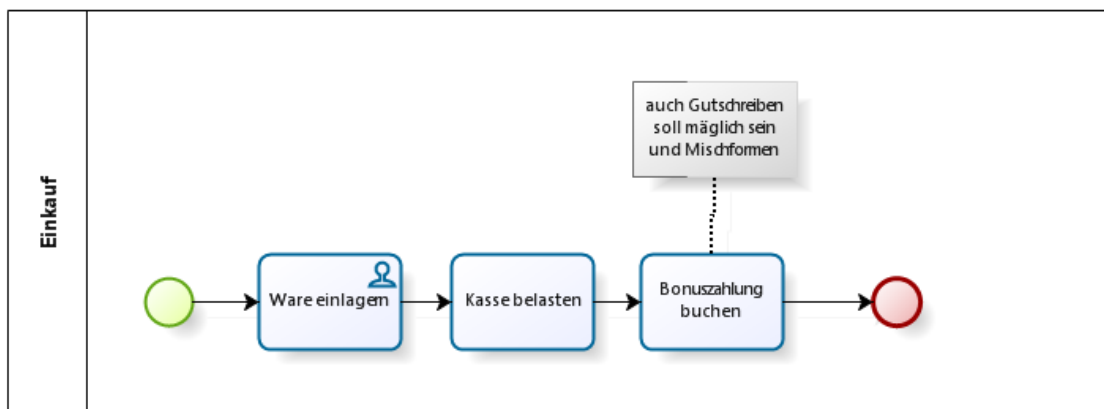


Abbildung 6.21.: Erster Entwurf: Einkaufsprozess

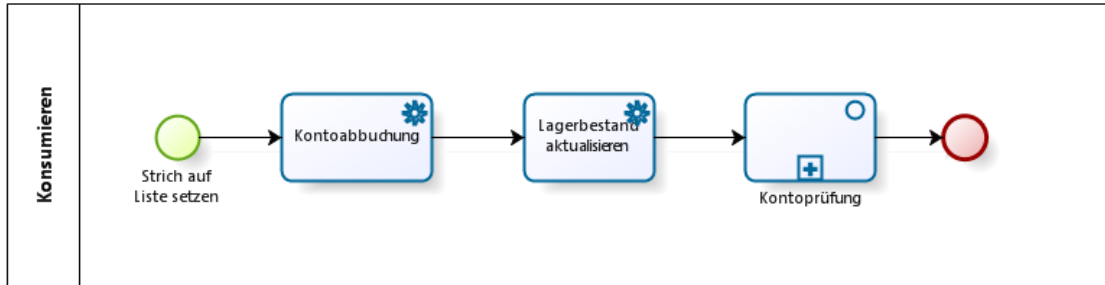


Abbildung 6.22.: Erster Entwurf: Konsumieren-Prozess

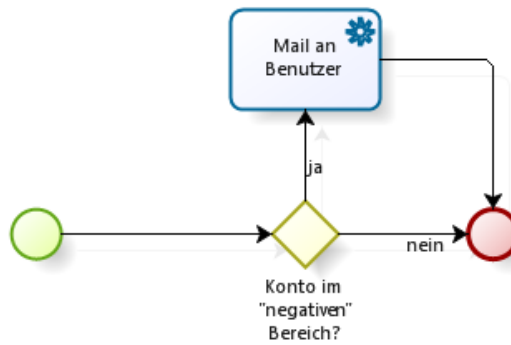


Abbildung 6.23.: Erster Entwurf: Kontopruefungsprozess

und vom Lagerbestand abgezogen. Die Kontobelastung wurde in den Abrechnungsprozess ausgelagert, der in „Accounting period“ umbenannt wurde, um zu verdeutlichen, dass der Lebenszyklus dieses Prozesses der Dauer eines Abrechnungszeitraums entspricht. Der überarbeitete „Accounting period“-Prozess (siehe Abbildung 6.25) beginnt nun mit dem Starten eines neuen Abrechnungszeitraums und dem Übernehmen eventueller Preisänderungen. Nach der Erzeugung der neuen Strichliste befindet der Prozess sich in einem Wartezustand, der erst verlassen wird, wenn der Verwalter den aktuellen Abrechnungszeitraum beendet. Im Anschluss wird automatisch ein neuer Abrechnungszeitraum und somit ein neuer „Accounting period“-Prozess gestartet, und der Verwalter hat die Möglichkeit, den über die Strichliste getätigten Getränkekonsum in die Anwendung zu übertragen.

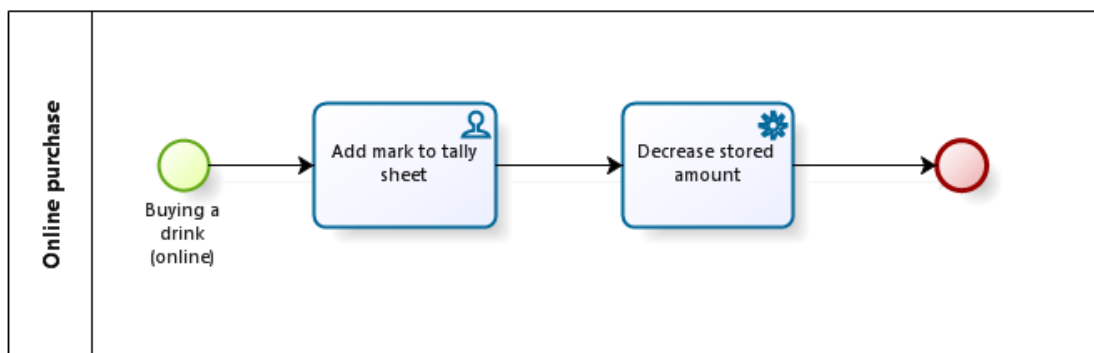


Abbildung 6.24.: Zweiter Entwurf: Konsumieren-Prozess

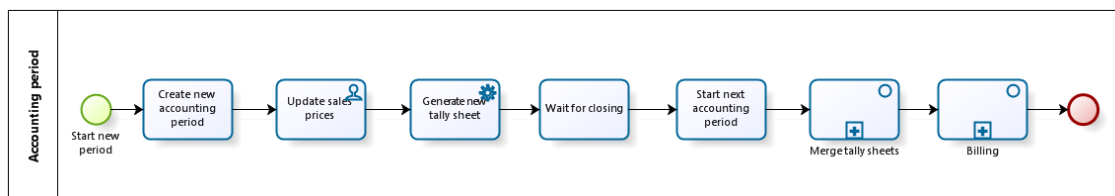


Abbildung 6.25.: Zweiter Entwurf: Abrechnungsprozess

Dabei wurden zwei neue Subprozesse identifiziert:

1. **Merge tally sheets** Dieser Prozess soll die über die Strichliste in die Anwendung eingetragenen Getränke mit den von den Anwendern online eingetragenen Getränke zusammenführen (siehe Abbildung 6.26).
2. **Billing** Dieser Prozess soll zunächst auf Basis des durch den „Merge tally sheets“-Prozesses zusammengeführten Getränkekonsums für jeden Anwender

den Betrag berechnen, mit dem das jeweilige Konto des Anwenders belastet werden soll. Im Anschluss erfolgt dann die Kontobelastung durch den Prozess (siehe Abbildung 6.27). Eine automatisierte Kontoprüfung mit anschließendem Versand von Rechnungen wurde in diesem Modellierungsschritt nicht mehr als sinnvoll erachtet. Vielmehr sollte dem Verwalter nun die Möglichkeit gegeben werden, sich jederzeit selbständig ein Bild über die aktuellen Kontostände zu verschaffen und bei Bedarf Rechnungen an die Anwender zu verschicken.

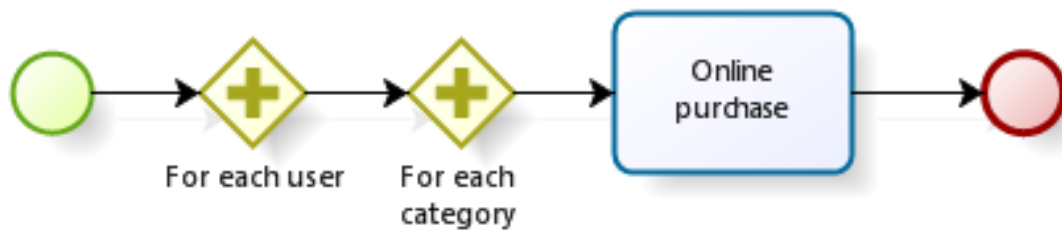


Abbildung 6.26.: Erster Entwurf: Merge-tally-sheets-Prozess

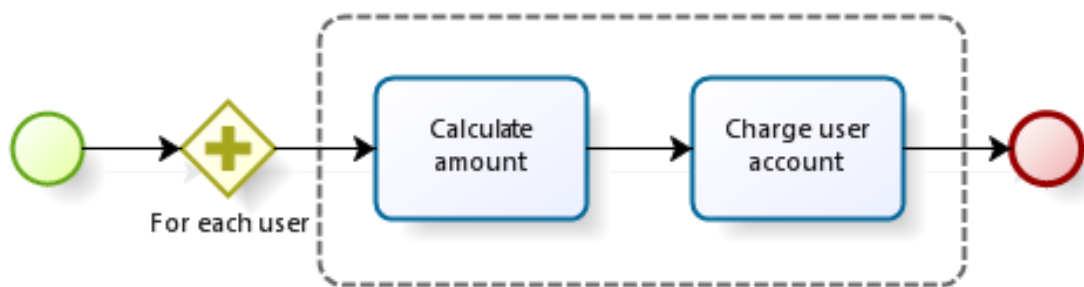


Abbildung 6.27.: Erster Entwurf: Billing-Prozess

Im Zuge der Überführung dieser Prozess-Skizzen in richtige jBPM-Prozesse, wurde in der Gruppe entschieden, dass der „Accounting period“-Prozess die Kontobelastung in zwei Schritten durchführen soll: Zunächst sollen nach dem Beenden eines Abrechnungszeitraums die in der Anwendung getätigten Getränkäufe sofort abgerechnet werden. Sobald der Verwalter anschließend die ausgehängte Strichliste in die Anwendung übertragen hat, werden auch die über die Strichliste getätigten Getränkäufe abgerechnet (siehe Abbildung 6.28).

Der Order-Prozess (zuvor Auftragsverteilung genannt) wurde nach Absprache mit dem Kunden im Zuge der Überführung in einen jBPM-Prozess ebenfalls ummodelliert. War zunächst vorgesehen, dass in einem ersten Schritt ein Einkaufsvorschlag

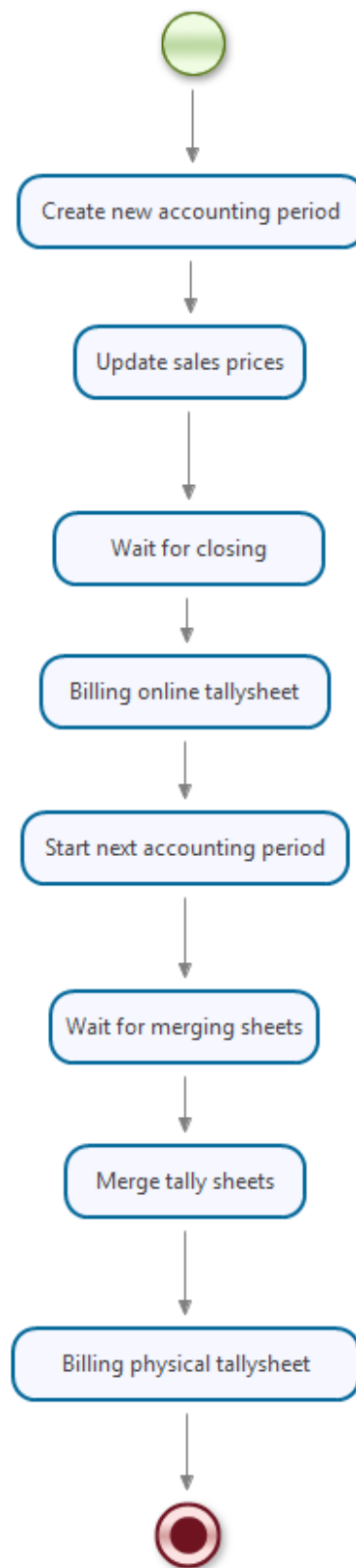


Abbildung 6.28.: jBPM-Prozess: Accounting period

erstellt und im zweiten Schritt der Einkäufer gewählt wird, so wird nun zunächst der Einkäufer ausgewählt und danach ein Einkaufsvorschlag angefordert. Der Grund für die Umkehrung der Reihenfolge ist der, dass der mögliche Einkaufsauftrag vom Volumen des Wagens des Einkäufers abhängen kann. Somit muss zunächst der Einkäufer bestimmt werden, bevor ein entsprechender Einkaufsauftrag generiert werden kann (siehe Abbildung 6.29).

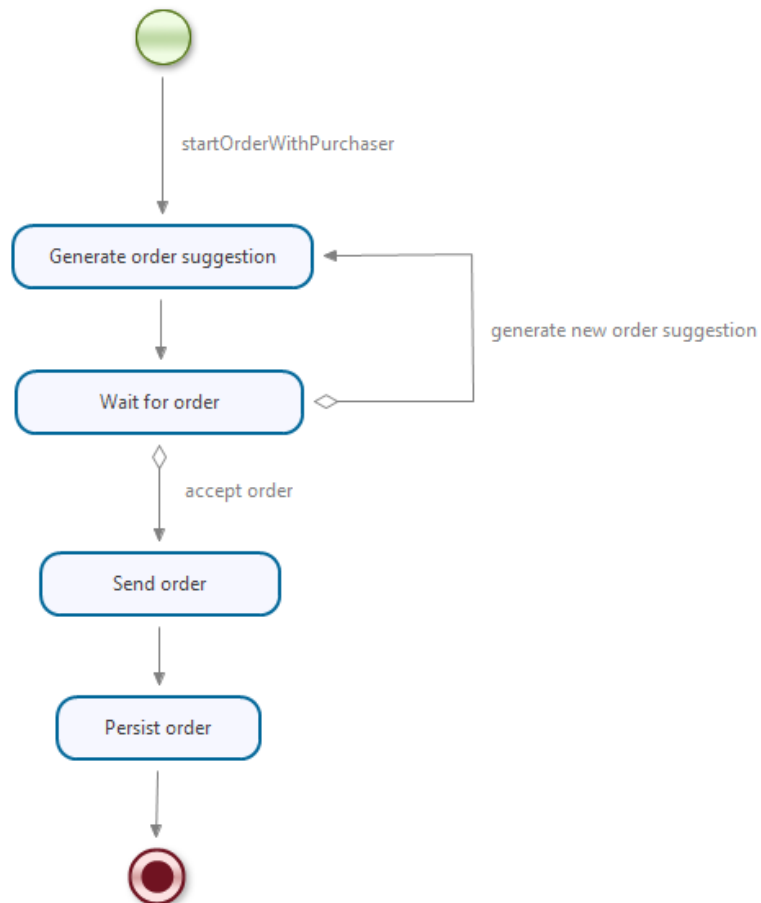


Abbildung 6.29.: jBPM-Prozess: Order

Der Einkaufsprozess wurde hingegen nicht als jBPM-Prozess implementiert, da darin nicht genug Geschäftslogik vorhanden war, die diesen Schritt gerechtfertigt hätte. Es handelt sich hierbei mehr um eine simple Aneinanderreihung von Service-methodeaufrufen, weshalb die Gruppe sich dazu entschieden hat, in diesem Fall auf einen jBPM-Prozess zu verzichten.

Die Prozesse „Merge tally sheets“ und „Billing“ wurden ebenfalls nicht als eigenständige jBPM-Prozesse implementiert, sondern tauchen in der endgültigen Version des „Accounting period“-Prozesses als Tasks dieses Prozesses auf.

Der dritte und letzte Prozess, der als jBPM-Prozess implementiert wurde, ist

somit der „Online purchase“-Prozess (siehe Abbildung 6.30). Dieser Prozess wird an zwei Stellen innerhalb der Anwendung aufgerufen:

- Beim Kauf eines Getränks durch den Anwender auf der Weboberfläche
- Beim Übertragen der Strichliste durch den Verwalter in die Anwendung

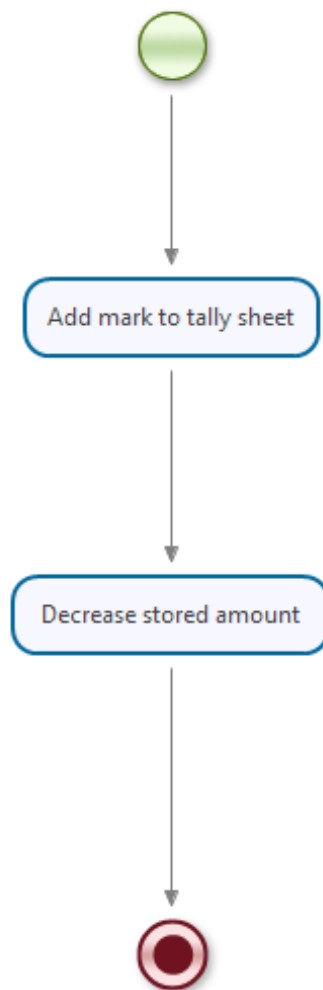


Abbildung 6.30.: jBPM-Prozess: Online purchase

**Verwendete Knotentypen** In den jBPM-Prozessen des dritten Projekts wurden hauptsächlich Knoten vom Typ *State* und *Custom* verwendet. Knoten vom Typ

*State* finden Verwendung, wenn die Prozesse auf eine Eingabe warten. Diese Knoten können leicht anhand ihrer Namen erkannt werden, da wir dafür durchgehend Bezeichnungen gewählt haben, die mit dem Wort „Wait“ beginnen, wie z.B. in „Wait for closing“. Die übrigen Knoten, die alle eine spezielle Aufgabe durchführen, sind vom Typ *Custom*. Wir haben uns für diesen Knotentyp entschieden, da er zum einen sehr einfach zu verwenden ist und zum anderen dem Programmierer große Freiheiten bei der Implementierung des Knotenverhaltens lässt. Das Knotenverhalten wird dabei durch eine Java-Klasse vorgegeben, die eine spezielle Methode implementieren muss, die bei Erreichen des Knotens aufgerufen wird. Somit lässt sich das Verhalten eines solchen Knotens recht einfach ändern, indem dem Knoten eine andere Java-Klasse zugeordnet wird.

#### 6.3.2.4. Serviceschicht

Die Serviceschicht des Projektes besteht aus den verschiedenen Geschäftsdiensten und bildet die Schnittstelle zwischen der Datenbankschicht und der Webschicht, bzw. der Prozessschicht. Da dieses Projekt mehr Logik enthielt, und deswegen auch mehr Methoden nach außen bekannt gibt, wurden die Geschäftsdienste in diesem Projekt thematisch aufgeteilt und in mehreren Services bekanntgegeben (siehe Abbildung 6.31). Insgesamt konnten dabei elf verschiedene Services gefunden werden, welche auch genau den Webservices entsprechen (siehe Abschnitt 6.3.2.5). Im Detail sind dies die folgenden Services:

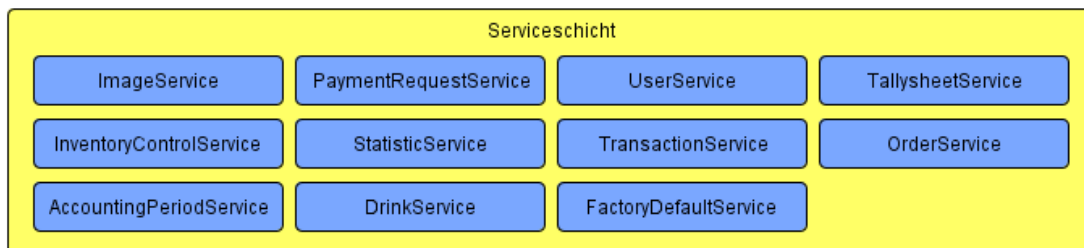


Abbildung 6.31.: Serviceschicht - Detailansicht

##### **AccountingPeriodService**

Der AccountingPeriodService bietet die Schnittstelle zu den Abrechnungszeitraum-Diensten. Über den Service legt die Prozessschicht Abrechnungszeiträume in der Datenbank an, und die Webschicht kann alle bekannten Abrechnungszeiträume abfragen.

##### **DrinkService**

Der DrinkService bietet Methoden an um neue Getränkekategorien, Getränke und Getränkepreise anzulegen, oder bestehende abzuändern. Zusätzlich liefert der Service bestehende Getränke zurück.



### **FactoryDefaultService**

Der `FactoryDefaultService` erzeugt die Standardrollen *yadoUser*, *yadoPurchaser* und *yadoAdmin*, sowie einen Administrator-Benutzer sofern die Rollen, bzw. der Benutzer nicht existieren. Er wird benutzt um die Anwendung zu initialisieren.

### **ImageService**

Der `ImageService` kann Bilder als BLOB in der Datenbank speichern und diese wieder als Bild zurückliefern.

### **InventoryControlService**

Der `InventoryControlService` speichert die Inventurdaten und erlaubt es diese abzuändern.

### **OrderService**

Der `OrderService` bietet Methoden um sowohl die Einkäufe als auch die Quittungen zu verwalten. Dazu können über diesen Einkäufe angelegt, bzw. abgefragt werden, sowie ebenfalls Quittungen erzeugt und abgefragt werden.

### **PaymentRequestService**

Der `PaymentRequestService` verwaltet die Zahlungsaufforderungen des Systems. Über diesen können Zahlungsaufforderungen für Benutzer erzeugt werden und alle bestehenden Zahlungsaufforderungen können angezeigt werden.

### **StatisticService**

Der `StatisticService` kann auf den in der Datenbank bestehenden Daten Statistiken berechnen, welche z.B. den Durchschnittsverbrauch eines Benutzers angeben.

### **TallySheetService**

Der `TallysheetService` bietet die Schnittstelle zu dem Onlineverbrauchszeitel. Beim Kauf über den Getränkeautomaten wird auf dieser Liste für den Benutzer und das Getränk ein virtueller Strich erzeugt. Die Anzahl der aktuell vorhandenen Striche auf der virtuellen Liste kann von diesem Service auch ausgelesen werden.

### **TransactionService**

Alle vorhandenen Buchungen des Systems werden vom `TransactionService` verwaltet. Dieser kann Buchungen erzeugen und wieder abfragen.

### **UserService**

Der `UserService` kann neue Benutzer und Rollen anlegen, sowie alle Benutzer des Systems zurückgeben.

Alle Services erben von einem Basis-Service, welcher Methoden zur Datenbank-Kommunikation bereitstellt (siehe Abbildung 6.32). Da es mehrere Services gibt,

die E-Mails versenden können sollen, stellt der Basis-Service dafür auch zusätzliche Methoden bereit. Wir haben ein Interface `IdHolder` eingeführt, welches jeder unserer EJBs implementiert und kennzeichnet, dass das EJB eine `id` vom Typ `long` besitzt. Die Methoden für die Datenbank-Kommunikation bestehen dabei aus

- **find(...)**: Eine Methode, die einen gegebenen `IdHolder` mit gegebener `Id` findet.

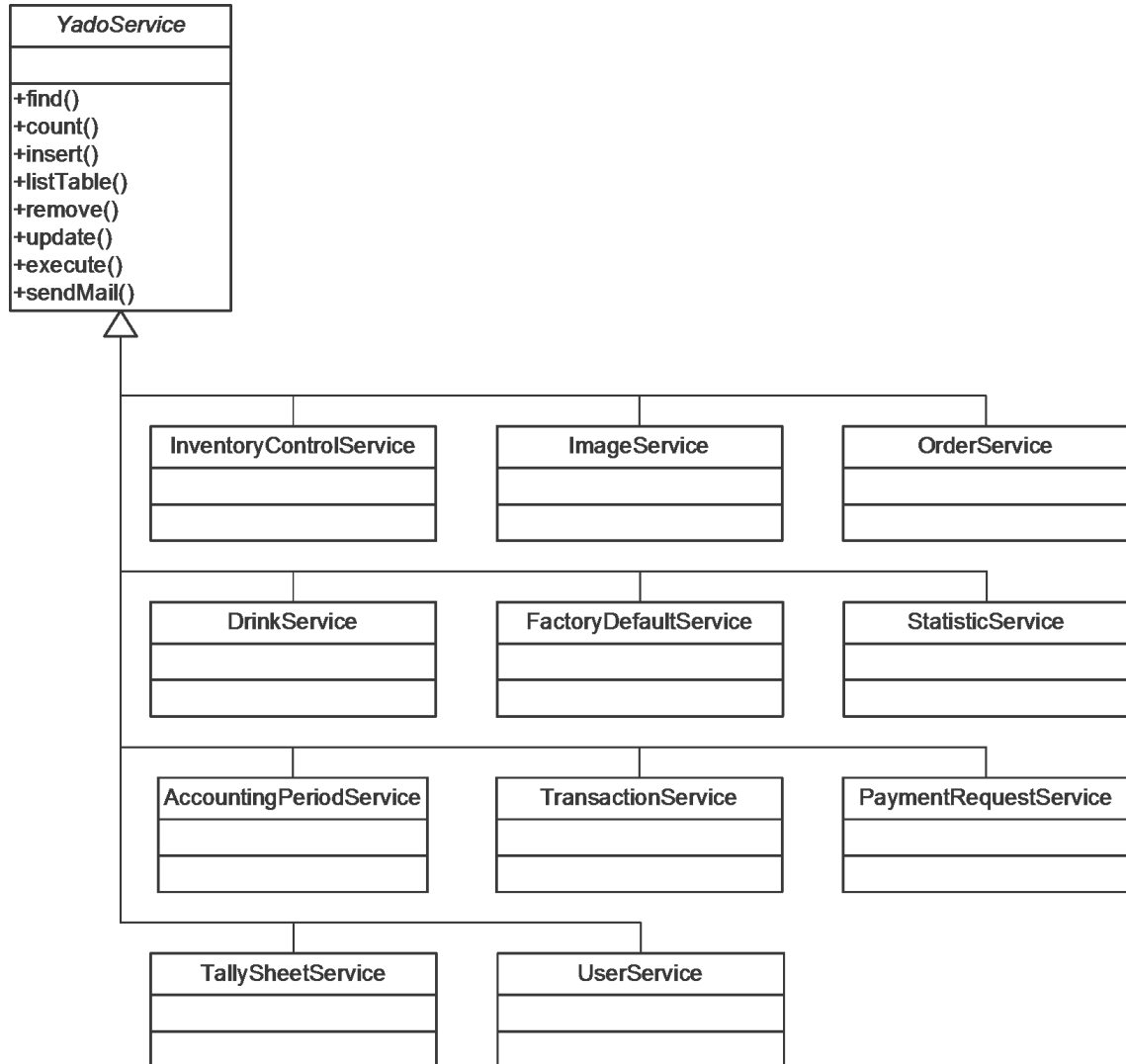


Abbildung 6.32.: Serviceschicht - Klassenhierarchie

- **count(...)**: Eine Methode, welche die Anzahl an Einträgen in der dem gegebenen `IdHolder` zugeordneten Tabelle findet.
- **insert(...)**: Eine Methode, welche einen `IdHolder` in die ihm zugeordnete Tabelle schreibt.
- **listTable(..)**: Eine Methode, welche die gesamte Tabelle eines `IdHolder`s ausliest.

- **remove(...)**: Eine Methode, welche einen IdHolder aus der Datenbank löscht.
- **update(...)**: Eine Methode, welche einen IdHolder in der Datenbank aktualisiert.
- **execute(...)**: Es gibt mehrere execute Methoden, welche Abfragen auf der Datenbank erlauben.

### 6.3.2.5. Webservices

Eine Anforderung an die Applikation war die Bereitstellung verschiedener Webservices. Die Anwendung sollte Schnittstellen zu anderen Clients bereitstellen können. Diese Clients sollen alle Funktionen der Anwendung nutzen können und sich komplett in das Programm integrieren. Zur Integration soll so wenig wie möglich am vorhandenen Programmcode geändert werden. Eine gute und standardisierte Methode zum Anbieten solcher Funktionen sind Webservices. Eine Recherche hat ergeben, dass die Integration von Webservices in Java EE Anwendungen recht gradlinig und automatisiert erfolgen kann. Es sollte die komplette Anwendung steuerbar sein, so dass die gleichen Schnittstellen, wie sie von der Präsentationsschicht genutzt werden, auch von den Webservices genutzt werden können. Eine Verteilung der Anwendung auf mehrere Systeme ist so weiterhin möglich.

**Wahl der Technologie** Da die Anforderung, unterschiedliche Clients an die Anwendung anzubinden, erst im Laufe des Projekts an Bedeutung gewann, hat sich die Gruppe dazu entschlossen, möglichst viel von schon vorhandenem Programmcode wiederzuverwenden.

Als Applikations- und Webserver wurde bislang der JBoss - vor allem wegen seiner Unterstützung von jBPM (siehe Kapitel 3.5) - genutzt. In der verwendeten Version bringt er von Haus aus auch eine Unterstützung für Webservices mit. Das JAX-WS Paket bringt in der installierten Version Unterstützung für Webservices mit [Oracle, 2011].

Über die Webservices sollen hauptsächlich Steuerbefehle mit der Anwendung ausgetauscht werden. Die Wahl der Webservice-Technologie fiel daher auf SOAP Webservices, die sich mit JAX-WS realisieren lassen.

Bei der Erstellung der WSDL-Daten hat sich die Wahl des Maven Build Systems bewährt. Die WSDL-Daten können mit dem JAX-WS Maven Plugin automatisch erstellt werden. Dabei ist aber darauf zu achten, dass alle Methodenaufrufe und Funktionsnamen in allen Klassen unterschiedlich benannt sind und keine Interfaces als Datentypen auftauchen.

**Lösungsmodell** Zur übersichtlichen Strukturierung des Projekts wurde ein weiteres Paket erstellt. Dieses soll komplett unabhängig von der bestehenden Präsentationsschicht sein. Das neue Modul soll die Webservices auf dem Applikationsserver

bereitstellen. Die einzelnen Funktionalitäten sollen allerdings weiterhin in den Service Modulen verbleiben. Die Erstellung des neuen Moduls hat zudem den Vorteil, dass der Anwender entscheiden kann, ob er die Webservicefunktionalität benötigt und das Modul deployt oder ob er auf das Modul verzichten möchte. Die Kommunikation des Moduls mit den vorhandenen Teilen findet wie bei der Präsentationsschicht per Remote Method Invocation (RMI) statt. Diese Kommunikation soll für den Benutzer des Moduls allerdings transparent sein, sodass er auf die Anwendung zugreift, als wäre es eine einzige zentrale Anwendung.

**Herausforderungen** Die gewählte JAX-WS Technologie kommt leider in der verwendeten Version nicht mit Java Interfaces zurecht. In den Service-Klassen wurde allerdings zur übersichtlichen Gestaltung und einfachen Implementierung über mehrere Pakete sehr viel Gebrauch von Java-Interfaces gemacht. Daher mussten Wrapper-Klassen für die unterschiedlichen Methoden erstellt werden. Eine einfache Wiederverwendung der implementierten Klassen der Interfaces war leider aufgrund von Konflikten in verschiedenen Hibernate-Annotationen nicht möglich.

Weitere Schwierigkeiten bereiteten die recht umfangreiche Standardinstallation von JBoss. Diese bringt viele Bibliotheken von Haus aus mit. Das Maven-JAX-WS-Tool bindet allerdings selbst auch einige Bibliotheken ein um nötige Abhängigkeiten abzudecken. Da es sich bei den Bibliotheken um verschiedene Versionen handelt, kommt es beim Deployen des Moduls zu Fehlern auf dem JBoss. Diese lassen sich allerdings durch einige zusätzliche Konfigurationen des Maven-Build-Skriptes beheben.

#### 6.3.2.6. Webschicht

Wie bei den vorherigen Projekten kam auch bei diesem Projekt wieder das Java-Framework Tapestry (s. Abschnitt 3.4) für die Webschicht zum Einsatz. Für den Login setzten wir nach wie vor auf JSP, da die Einbindung eines auf JAAS gestützten Authentifizierungs- und Authorisierungsdienstes in Tapestry für uns nicht so einfach realisierbar war.

Zunächst wird die Webanwendung funktional beschrieben, sie unterscheidet grob fünf Bereiche:

1. **Getränkeautomat** (für jeden Benutzer zugänglich)  
Hier besitzt jeder Benutzer die Möglichkeit, ein im System registriertes Getränk auszuchecken. Anstatt also einen Strich auf der Strichliste in Papierform zu machen, kann hier jeder Benutzer einen virtuellen Strich vornehmen (s. Abbildung 6.33), woraufhin der hinterlegte Getränkepreis sofort verrechnet und auf dem Kontostand des Benutzers berücksichtigt wird.
2. **Mein Konto** (für jeden Benutzer zugänglich)  
Sämtliche benutzerspezifischen Aktionen sind unter diesem Menüpunkt vereint. (s. Abbildung 6.34)

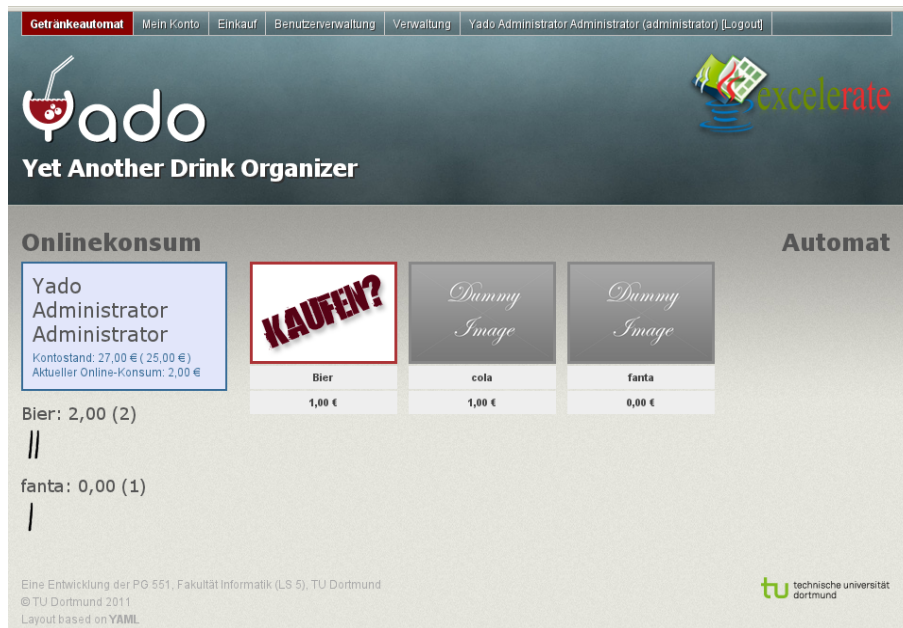


Abbildung 6.33.: Getränkeautomat

- **Kontoübersicht**  
Hier wird eine Übersicht über das eigene Konto geboten, wo sämtliche Transaktionen über einen frei wählbaren Zeitraum aufgelistet werden können.
  - **Passwort ändern**  
Hier wird dem Benutzer die Möglichkeit gegeben sein Passwort zu ändern. Dazu muss der Benutzer zur Bestätigung sein aktuelles Passwort ebenfalls mit angeben.
  - **Verbrauchsübersicht**  
Diese zeigt dem Benutzer den eigenen Getränkeverbrauch tabellarisch und als Balkendiagramm.
  - **Einkaufsübersicht (für Einkäufer zugänglich)**  
Besitzt der Benutzer die Rolle des Einkäufers, so kann sich dieser hier die eigenen bereits getätigten Einkäufe anzeigen lassen.
3. **Einkauf** (für Verwalter zugänglich)  
Die Aktionen, welche die Einkäufe betreffen, sind unter diesem Menüpunkt zusammengefasst. Hier gibt es die Möglichkeiten:
- **Einkauf abrechnen**  
Wurde ein Einkauf vorgenommen, so kann ein Verwalter hier diesen Einkauf ins System übertragen. Es kann ein vorher erstellter Einkaufsauftrag ausgewählt werden, falls dieser existiert und der Einkäufer, der diesen Auftrag tatsächlich ausgeführt hat. Daraufhin kann der ursprüngliche



Abbildung 6.34.: Mein Konto

Auftrag angepasst werden, da es nicht gegeben sein muss, dass der tatsächliche Einkauf dem ursprünglichen Auftrag 1 zu 1 entspricht.

- Einkauf beauftragen  
Unter diesem Menüpunkt können neue Einkaufsaufträge vergeben werden. Der gewählte Einkäufer wird per E-Mail über den Auftrag informiert.

#### 4. **Benutzerverwaltung** (für Verwalter zugänglich)

Sämtliche Verwaltungsoptionen, die Benutzer betreffen, wurden unter einem eigenen Menüpunkt zusammengefasst. Dieser bietet:

- Benutzer anlegen/bearbeiten
- Benutzerübersicht  
Eine Auflistung sämtlicher im System registrierten Benutzer (s. Abbildung 6.35), inklusive der Anzeige welchen Rollen dieser angehört und der Möglichkeit diesen Benutzer zu bearbeiten.
- Kontenübersicht  
Eine Übersicht aller registrierten Benutzer und deren aktuellen Kontoständen und der Möglichkeit Zahlungsaufforderungen zu versenden. Sollte sich der Kontostand eines Benutzers unterhalb von -10 EUR befinden, so wird dieser für den Versand einer Zahlungsaufforderung vorselektiert.
- Buchungen  
Eine Buchungsübersicht, welche sämtliche Transaktionen für einen gewählten Benutzer anzeigt.

- **Rechnungshistorie**  
Hier können sämtliche versendeten Zahlungsaufforderungen für einen gewählten Benutzer angezeigt werden.
- **Einzahlen**  
Hierunter verbirgt sich ein Formular, worüber eine Einzahlung für einen Benutzer vorgenommen werden kann. Zusätzlich zu dem Betrag kann ein freier Text (z.B. Bareinzahlung) als Kommentar mit angegeben werden.
- **Rundmail**  
Hier wird Verwaltern die Möglichkeit geboten eine Mail an alle Benutzer, Einkäufer oder Verwalter zu schicken.

Benutzerübersicht

Benutzername	Vorname	Nachname	E-Mail	Aktiv	Einkäufer	Verwalter	
administrator	Yado Administrator	Administrator	nemail@example.com	✓	✓	✓	Bearbeiten
dani	daniil	adoua	ettiboa@hotmail.com	✓	✓	✓	Bearbeiten

Eine Entwicklung der PG 551, Fakultät Informatik (LS 5), TU Dortmund  
© TU Dortmund 2011  
Layout based on YAML

tu technische universität dortmund

Abbildung 6.35.: Benutzerübersicht

## 5. Verwaltung (für Verwalter zugänglich)

- **Lagerverwaltung**  
Die Lagerverwaltung bietet neben einer Übersicht aller im Lager befindlichen Getränke die Möglichkeit den Lagerbestand anzupassen, falls dieser vom tatsächlichen Bestand abweichen sollte.
- **Abrechnung**
  - **Übersicht**  
Hier gibt es eine Übersicht über alle konsumierten Getränke innerhalb gewählter Grenzen von Abrechnungszeiträumen.
  - **Abrechnungszeitraum beenden**  
Soll ein Abrechnungszeitraum beendet werden, so kann dies hier vorgenommen werden. Die Anwendung bietet hier noch einmal die Möglichkeit Getränkepreise anzupassen oder Getränkekategorien oder Ge-

tränkesorten hinzuzufügen oder zu ändern. Gibt es noch keinen laufenden Abrechnungszeitraum, so kann der initiale Abrechnungszeitraum ebenfalls unter diesem Menüpunkt gestartet werden. Wird ein laufender Abrechnungszeitraum beendet, dann stellt die Anwendung hier ein Formular zur Verfügung, wo sämtliche Striche der Strichliste in Papierform in die Anwendung übertragen werden können.

- Liste herunterladen  
Dieser Menüpunkt bietet die Möglichkeit eine Strichliste aus den im System registrierten Benutzern und Getränken zu generieren und diese als PDF-Datei herunterzuladen. die PDF-Datei kann anschließend ausgedruckt und im Getränkelager aufgehängt werden.
- Getränke
  - Getränkekategorien
  - Getränkesorten
  - Preisänderungen  
Unter diesem Menüpunkt können Preisänderungen einer Getränkekategorie vorgenommen werden. Diese Preisänderungen werden beim Starten eines neuen Abrechnungszeitraums übernommen.
- Kasse  
Hierunter verbirgt sich die Verwaltung der Kasse. Neben einer Übersicht aller getätigten Buchungen über einen gewählten Zeitraum gibt es hier die Möglichkeit den Kassenstand direkt zu ändern und diese Änderung mit einem Kommentar zu versehen.
- Anonymer Getränkekauf  
Gemäß Userstory 5 der Priorität 2 besteht hier für Verwalter die Möglichkeit anonyme Getränkekäufe mit der Anwendung zu erfassen. Überschüssige Geldbeträge werden als gesonderte Buchung *Spende* im System berücksichtigt.
- Statistik
  - Monatliche Einzahlungsübersicht  
Eine tabellarische Übersicht aller Benutzer und deren durchschnittlichen Einzahlungen auf Monatsbasis umgerechnet.
  - Getränke-Rangliste (s. Abbildung 6.36)  
Hier wird Verwaltern in einem Barchart angezeigt wieviel Getränke welcher Kategorie in einem gewählten Zeitraum konsumiert wurden. Das Barchart ist stets absteigend sortiert, so dass die beliebtesten Getränke oben stehen.
  - Durchschnittsverbrauch  
Mit Hilfe dieser Statistikseite wird Verwaltern die Möglichkeit gegeben zu ermitteln, welche Getränke durchschnittlich je Tag, Woche,



Monat oder Jahr konsumiert werden. Dabei werden die tatsächlich konsumierten Getränke gleichmäßig auf die gewählte Einheit um- bzw. hochgerechnet.



Abbildung 6.36.: Getränke-Rangliste

**Login** Wie zu Beginn dieses Abschnittes (s. 6.3.2.6) bereits erwähnt, wurde für den Login auf JAAS gesetzt. Die Einbindung von JAAS in Tapestry gestaltete sich allerdings also so schwierig für uns, so dass wir an dieser Stelle die Tatsache ausgenutzt haben, dass eine Session zwischen JSP und Tapestry geteilt wird. So war es uns möglich den Login mit Hilfe eines Formulars in JSP durchzuführen.

**Services** Während der ersten beiden Projekte haben wir gemerkt, dass es während der parallelen Entwicklung der Service- und Webschicht von Vorteil ist die angebotenen Services innerhalb der Webschicht noch einmal zu kapseln. Wir hatten des öfteren Probleme mit Exceptions, welche in der Service-Schicht auftraten und die Entwicklung der Webschicht ausgebremst haben. Da bei der Entwicklung der Webschicht zunächst die Templates funktional erstellt werden mussten und solche Fehler dazu geführt haben, dass die Webanwendung die Templates meist erst gar nicht dargestellt hat, wurden zunächst leere Services innerhalb der Webschicht implementiert, welche lediglich die benötigten Methoden bereitstellten, aber noch nicht an die Service-Schicht angebunden waren. So war eine weitestgehend unabhängige Entwicklung der Webschicht möglich. Später wurden diese Services dann mit der

Service-Schicht verbunden, wobei allerdings darauf geachtet wurde, dass sämtliche Exceptions, die eventuell innerhalb der Service-Schicht auftreten können, innerhalb der Services der Webschicht abgefangen und geloggt wurden. Durch diese Maßnahme wurde die Entwicklung auf der Seite der Webschicht deutlich beschleunigt, da wir uns somit nicht immer direkt mit den aufgetretenen Fehlern beschäftigen mussten.

### 6.3.3. Zusammenfassung

Das dritte Projekt war das größte und wichtigste Projekt der PG. Es umfasste das gesamte zweite PG-Semester.

Das Ziel dieses Projektes war es, anhand der gegebenen Excel-Datei (“Verwaltungsdokument”) eine Getränkeverwaltung zu entwickeln, die mit Hilfe eines Browsers bedienbar ist und alle Funktionen der Excel-Datei umfasst. Dieses Ziel hat die PG erreicht. Die entwickelte Anwendung enthält alle durch das “Verwaltungsdokument” abgebildeten Funktionen zur Kundenkonto- und Lagerverwaltung und erleichtert darüber hinaus auch die Kundenverwaltung. Für das Versenden eines Einkaufsvorschlags musste der Verwalter bisher die benötigten Getränke selbst bestimmen und von Hand in eine E-Mail-Vorlage eintragen. Dies ist jetzt auch innerhalb der Anwendung möglich. Auf Wunsch kann das System sogar das Kaufverhalten der Kunden auswerten und eine Verbrauchsvorhersage treffen, auf Basis dieser der Verwalter dann einen Einkaufsauftrag erstellen kann. Auch weitere Statistiken, wie z.B. eine Getränkeangliste (d.h. eine Übersicht der am häufigsten verkauften Getränke) ist verfügbar. Außerdem haben Kunden jetzt die Möglichkeit, sich selbst beim System einzuloggen, ihren derzeitigen Kontostand einzusehen und über den Online-Getränkeautomaten unabhängig von der am Kühlschrank aushängenden Strichliste Getränke zu kaufen.

Durch die Anbindung der Webservice-Schnittstelle, mit der sowohl Funktionen der Serviceschicht als auch der Prozessschicht angesprochen werden können, ist es möglich das Projekt auch von außen mit weiterer Software zu ergänzen, ohne den Kern des Programms anfassen zu müssen. Die neue Anwendung ist also nicht nur einfacher zu bedienen, sondern enthält auch zahlreiche weitere Funktionen, die die bisherige Lösung nicht bereitstellte.

Durch den Projektleiter, der von Anfang an die Übersicht behielt, und die regelmäßigen Treffen (besonders die nicht offiziellen), konnte das Projekt wie geplant umgesetzt werden, und es traten nur selten Verzögerungen auf. Die Aufgabenverteilung erfolgte während des gesamten Projektes über das webbasierte Projektmanagement-Tool Redmine [Lang] (s. Abschnitt 4.4), was insgesamt zu einer gleichmäßigeren Verteilung der Arbeit auf alle Mitglieder führte, als es in den anderen Projekten der Fall war. Die Webservice-Schnittstelle wurde rechtzeitig fertiggestellt, obwohl sie nicht von Anfang an Teil der Anforderungen des Kunden an das neue System zur Getränkeverwaltung war. Die frühe Identifizierung der Geschäftsprozesse, die dem “Verwaltungsdokument“ zugrunde liegen, half der PG, die Funktionsweise der Anwendung besser zu verstehen, und die Kernfunktionalität zu extrahieren. Im Gegensatz zum zweiten Projekt (“Stundenzettel“) wurde das Datenmodell nach und

nach entworfen. Dabei konzentrierte sich die PG zunächst auf die hochpriorisierten Userstories und die restlichen Teile des Datenmodells wurden nur grob formuliert. Dadurch waren die zu implementierenden Funktionen von Woche zu Woche überschaubarer, und Fehler und Lücken im Datenmodell fielen früher auf, weil fast die gesamte PG am gleichen Teil der Anwendung arbeitete. So konnte die PG auch das letzte Projekt erfolgreich abschließen.



## 7. Kritik

Zu Beginn der Projektgruppe ist aufgefallen, dass trotz des Starts der regelmäßigen wöchentlichen Treffen die Kommunikation noch ausbaufähig war. Es fanden noch keine Treffen der PG-Teilnehmer außerhalb der wöchentlichen Besprechung statt und auch Medien wie Skype, ICQ u.ä. wurden nicht von allen zur Kommunikation in Anspruch genommen. Im Laufe der ersten Semesterwochen hat sich dies verbessert - es wurden alle Kontaktdaten ins BSCW eingetragen und zwei Tage bestimmt, an denen sich die PG in den Poolräumen trifft. Zudem wurde sich auf ICQ als Tool zur Kommunikation geeinigt. Dieses wurde von vielen Leuten auch benutzt, allerdings waren bis zum Ende der PG einige Teilnehmer doch nur sporadisch darüber erreichbar, so dass sich letztendlich E-Mail als Kommunikation durchgesetzt hat. Über diese waren alle erreichbar, und bei größeren Absprachen, welche einen längeren Dialog benötigten, hat man sich dann doch wieder im ICQ getroffen.

Bei der Umsetzung der Projekte wurde während des ersten Semesters erst sehr spät ein Projektleiter bestimmt, was bis dahin manchmal zu mangelndem Überblick führte und unzureichender Koordination. So kam es, dass z.B. die Arbeitsverteilung ungleichmäßig war und Leerläufe bei einigen Teilnehmern entstanden, die durch Parallelisierung der Aufgaben durch einen Projektleiter hätten vermieden werden können. Im Laufe des zweiten Projektes wurde dann ein Projektleiter bestimmt, welcher alle Aufgaben verteilt hat und immer die Übersicht über alle anstehenden Aufgaben hatte. Dadurch konnte die Produktivität der PG enorm gesteigert werden, was sich insbesondere Anfang bis Mitte des dritten Projekts gezeigt hat. Durch den Projektleiter und seine Aufgabenverteilung haben sich auch auf bestimmte Themen spezialisierte kleine Gruppen gebildet, welche direkt untereinander kommuniziert haben.

Im Laufe des ersten Semesters ist aufgefallen, dass nicht alles auf Anhieb optimal gelöst worden ist. Unter anderem wurde durch die Projektgruppe kritisiert, dass die Vielzahl an Tools dazu führte, dass die Handhabung unübersichtlich wurde. Nachdem neben BSCW und Origo noch der PivotalTracker eingeführt wurde, fiel es einigen Teilnehmern schwer, alle Tools zu pflegen und auf dem aktuellsten Stand zu halten. Nachdem BSCW schon von Beginn der Projektgruppe an genutzt wurde um die Seminarreihe zu organisieren und die Kontakte zu verwalten, kam später noch Origo dazu, da eine Wiki-Struktur und ein SVN-Repository vonnöten waren. Da in beiden Tools keine ausreichende BackLog-Funktion vorhanden ist, wurde der PivotalTracker (später ausgetauscht durch Redmine) als drittes Tool hinzugenommen. Zu dem Zeitpunkt war es aber nicht mehr möglich eines der vorherigen Tools abzu-

stoßen. Doch wurde von Origo zum Schluss nur noch das SVN-Repository benutzt, und das Wiki ist immer mehr in den Hintergrund gewandert.

Ein weiterer Kritikpunkt ist das Umsetzen der Aufgabe anhand von Schichten, die zu Beginn des zweiten Projekts nicht mit dem Schichtenmodell aus dem ersten Projekt übereinstimmten. Zunächst wurden nur wenige Prozesse identifiziert. Die PG hat aber im Nachhinein erkannt, dass doch mehr Prozesse als angenommen involviert sind. Bei wenigen Prozessen wäre ein Vorgehen ohne Einbeziehen einer Prozessschicht denkbar, bei der Vielzahl an Prozessen, die im Nachhinein jedoch identifiziert wurden, war es dann doch vonnöten eine Prozessschicht in das benutzte Schichtenmodell zu integrieren, um so die Logik aus der Serviceschicht in die Prozessschicht auszulagern. Würde man dies nicht tun, könnte man sonst später vor dem Problem stehen, dass geringfügige Änderungen mit einem erheblichen Aufwand für Änderungen in der Serviceschicht verbunden wären, was bei Verwenden einer Prozessschicht nicht der Fall wäre. Im dritten Projekt wurden die Prozesse von Anfang an identifiziert, so dass die Verwendung einer Prozessschicht von Anfang an geplant wurde, und nicht später noch hinzugefügt werden musste.

Am Anfang der PG wurde entschieden, dass wir JBoss als Applikations-Server einsetzen. Bis auf jBPM hätten alle benötigten Technologien auch mit anderen Applikations-Servern, wie z.B. Glassfish oder sogar einem einfachen Tomcat, benutzt werden können. Da jBPM allerdings im JBoss bereits fertig implementiert ist, und somit der Einsatz von jBPM bei Benutzung des JBoss erleichtert ist, haben wir uns für den JBoss als Applikations-Server entschieden. Dies stellte sich während der PG manchmal als Nachteil heraus, da wir oft Probleme hatten verschiedene Technologien auf dem JBoss vernünftig zu konfigurieren. Bei Einsatz einer neuen Technologie oder Neukonfigurierung dieser, führte es auch dazu, dass es immer wieder dazu kam, dass jemand seine Konfigurierung änderte, und dies dann per E-Mail kommuniziert werden musste, damit jeder PG-Teilnehmer diese Änderung übernimmt. Manchmal gab es dabei Kommunikationsschwierigkeiten, so dass manche Teilnehmer bis zum nächsten Treffen nicht arbeiten konnten. Zusätzlich muss leider gesagt werden, dass der JBoss sehr Hardware hungrig ist, und manchmal selbst stärkere Maschinen in die Knie gezwungen hat.

## 8. Zusammenfassung

Als Resumée des ersten Semesters kann man hinsichtlich der vorgegebenen Minimalziele (siehe Abschnitt 2.2) sagen, dass diese innerhalb der Projekte des Semesters erreicht wurden.

Die Lagerverwaltung wurde erfolgreich implementiert und war - bis auf geringfügige zeitbedingte Abstriche, die mit dem Kunden abgesprochen waren - lauffähig. Die Abstriche, die gemacht wurden, haben sich jedoch nicht negativ auf das Lernziel dieses Projekts ausgewirkt, sondern befanden sich nur im Bereich der grafischen Darstellung und in Funktionen, für deren Umsetzung das Wissen im Verlauf des Projekts bereits angeeignet wurde und lediglich auf die Umsetzung aus Zeitgründen verzichtet wurde. Anhand des ersten Projekts wurde die dafür aufgebaute Entwicklungsumgebung auf die Probe gestellt und konnte somit als Basis für das zweite Projekt dienen.

Zusätzlich zur bereits aufgebauten Entwicklungsumgebung war auch die Benutzerverwaltung schon als Basis für das Projekt „Stundenzettel“ während des ersten Projekts erarbeitet worden. Im zweiten Projekt wurde erfolgreich eine Stundenzettelverwaltung durch Bearbeitung per Webinterface erleichtert und somit ein Grundbaustein für die Bearbeitung und Verwaltung komplexerer Excel-Dokumente gelegt.

Im zweiten Semester konnten wir uns mit dem im ersten Semester gewonnenen Wissen dann endgültig auf ein Projekt konzentrieren. Da wir für das dritte Projekt auch einen Projektleiter hatten, der den Überblick behalten konnte, konnte das Projekt fast ohne Leerlauf oder Verzögerungen durchgeführt werden. Dies führte dann auch dazu, dass diesmal alle Prioritäten durchgeführt werden konnten. Der strukturelle Aufbau des dritten Projektes war auch sehr ähnlich zum zweiten Projekt, was eine Einarbeitung in die Architektur auch sehr gering hielt.

Das erste Projekt wurde zeitlich wie geplant abgeschlossen, das zweite hat dagegen ein paar Wochen mehr Zeit in Anspruch genommen als ursprünglich geplant. Dies ist unseres Ermessens nach vor allem auf die langwierige Phase des Aufbaus der Entwicklungsumgebung und die lange Zeit fehlende Projektleitung zurückzuführen. Trotz der Kritikpunkte kann man zusammenfassend sagen, dass das erste Semester positiv verlaufen ist. Das zweite Semester konnte, u.a. auch dank der eingeführten Projektleitung, auch wie geplant abgeschlossen werden. Eine Phase des Aufbaus gab es nicht, so dass wir uns voll auf die gestellte Aufgabe konzentrieren konnten. Rückblickend ist das zweite Semester unserer Meinung nach ebenfalls positiv verlaufen.

# Anhang

## A. Administratorhandbuch





**YaDO - Administrator Handbuch  
PG 551: "Excelerate"**

**Ettiboa Adouakou, Georgios Christidis, Alexander Drees, Felix Gabriel, Christian Hammerl, Kersten Lorenz, Donato Marro, Jan Pardo, Carola Thalmann, Julia Werdelmann, Hongzhi Wang**

21. September 2011



**Institution**

Technische Universität Dortmund  
Fakultät für Informatik  
Lehrstuhl 5 für Programmiersysteme

**Betreuer**

Dipl.-Inform. Markus Doedt  
Dipl.-Inform. Sven Jörges



# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
<b>2</b>	<b>JBOSS</b>	<b>3</b>
2.1	Download . . . . .	3
2.2	Java . . . . .	3
2.3	Konfigurieren . . . . .	3
2.3.1	Startdatei . . . . .	3
2.3.2	HSQL-DB im JBOSS . . . . .	4
2.3.3	JAAS-Konfiguration . . . . .	5
2.4	Erster Start . . . . .	6
<b>3</b>	<b>JBPM</b>	<b>7</b>
3.1	Download . . . . .	7
3.2	Installation . . . . .	7
3.2.1	HSQL-Datenbank . . . . .	7
3.3	Testen der Datenbank . . . . .	8
3.4	jBPM im JBOSS . . . . .	8
<b>4</b>	<b>YaDO</b>	<b>9</b>
4.1	Grundlegendes . . . . .	9
4.2	Erster Start . . . . .	9
4.3	Webservices . . . . .	10



# 1 Einleitung

Dies ist das Administrator Handbuch der YaDO-Anwendung, welche im Rahmen der Projektgruppe 551 'Excelerate' an der TU-Dortmund entwickelt wurde. Es beschreibt wie man die Anwendung installiert. Dazu gehören unter anderem die Installation und Konfiguration des JBoss Anwendungsservers, die Installation von jBPM und die Einrichtung der HSQLDB-Datenbank.

Anschliessend wird noch kurz auf die Inbetriebnahme von YaDO eingegangen.



## 2 JBOSS

Für die Installation von YaDO wird ein Projektverzeichnis, hier `yado`, benötigt.

### 2.1 Download

Als erstes muss der JBOSS mit der Version 5.1.0 GA heruntergeladen werden<sup>1</sup>. Die Zip-Datei wird in den Ordner `yado/jboss/jboss-5.1.0.GA` entpackt. Der gesamte Pfad muss dann als Umgebungsvariable `JBOSS_HOME` gesetzt werden.

### 2.2 Java

JBoss 5.1 läuft sowohl unter Java 5 als auch Java 6. Lässt man den Server unter Java 6 laufen, muss man allerdings vorher noch folgende Libraries aus dem `JBOSS_HOME/client` Verzeichnis nach `JBOSS_HOME/lib/endorsed` kopieren:

- `jbossws-native-saaj.jar`
- `jbossws-native-jaxrpc.jar`
- `jbossws-native-jaxws.jar`
- `jbossws-native-jaxws-ext.jar`

### 2.3 Konfigurieren

#### 2.3.1 Startdatei

Die Startdatei muss konfiguriert werden um dem JBoss mehr Speicher zur Verfügung zu stellen. Dazu wird die `run.sh` (bei Linux), bzw die `run.bat` (bei Windows) angepasst werden indem man die `JAVA_OPTS` wie unten angegeben anpasst.

Linux:

```
JAVA_OPTS="-XX:MaxPermSize=256m -Djboss.vfs.forceCopy=false "
```

Windows:

---

<sup>1</sup><http://www.jboss.org/jbossas/downloads/>



```
set JAVA_OPTS="-XX:MaxPermSize=256m -Djboss.vfs.forceCopy=false "
```

Die ersten Zeilen sollten dann so aussehen:

Linux:

```
#!/bin/sh
### ===== ###
## ##
## JBoss Bootstrap Script ##
## ##
### ===== ###

### $Id: run.sh 88978 2009-05-16 18:18:45Z $ ###

DIRNAME='dirname xx-tagmarker2-xx'
PROGRAMME='basename xx-tagmarker2-xx'
GREP="grep"

JAVA_OPTS="-XX:MaxPermSize=256m -Djboss.vfs.forceCopy=false "

# Use the maximum available, or set MAX_FD != -1 to use that
MAX_FD="maximum"

...
```

Windows:

```
@echo off
rem -----
rem JBoss Bootstrap Script for Windows
rem -----

rem $Id: run.bat 88978 2009-05-16 18:18:45Z $

set "JAVA_OPTS=-XX:MaxPermSize=256m -Djboss.vfs.forceCopy=false "

@if not "%ECHO%" == "" echo %ECHO%
...
```

### 2.3.2 HSQl-DB im JBOSS

Es muss die DataSource eingetragen werden mit der wir später verbinden. Dazu fügt man in der Datei `hsqldb-ds.xml` im `deploy`-Verzeichnis des JBoss folgendes hinzu:

```

<local-tx-datasource>
  <jndi-name>pg551DS</jndi-name>
  <connection-url>jdbc:hsqldb:hsqldb://localhost:1701</connection-url>
  <driver-class>org.hsqldb.jdbcDriver</driver-class>
  <user-name>sa</user-name>
  <password></password>
  <min-pool-size>1</min-pool-size>
  <max-pool-size>5</max-pool-size>
  <idle-timeout-minutes>0</idle-timeout-minutes>
  <track-statements />
  <prepared-statement-cache-size>32</prepared-statement-cache-size>
  <metadata>
    <type-mapping>Hypersonic SQL</type-mapping>
  </metadata>
</local-tx-datasource>

```

In der Datei `yado/jboss-5.1.0.GA/server/default/conf/jbossts-properties.xml` des JBoss muss folgendes ergänzt werden, um mehrere Verbindungen zur Datenbank zu erlauben:

```

<properties depends="arjuna" name="jta">
  ...
  <property name="com.arjuna.ats.jta.allowMultipleLastResources" value="true"/>
</properties>

```

### 2.3.3 JAAS-Konfiguration

In der Datei `yado/jboss-5.1.0.GA/server/default/conf/login-config.xml` muss eine application-policy namens `pg551-domain` eingetragen werden:

```

<application-policy name="pg551-domain">
  <authentication>
    <login-module code="org.jboss.security.auth.spi.
      DatabaseServerLoginModule" flag="required">
      <module-option name="dsJndiName">java:/pg551DS</module-option>
      <module-option name="principalsQuery">
        SELECT password FROM yado_user where login=?
      </module-option>
      <module-option name="rolesQuery">
        SELECT t3.rolename, 'Roles' FROM yado_user t1, yado_usersroles
          t2, yado_roles t3 WHERE t1.login=? AND t1.id = t2.userid
          AND t2.roleid=t3.id
      </module-option>
      <module-option name="hashAlgorithm">MD5</module-option>
    </login-module>
  </authentication>
</application-policy>

```

```
</authentication>  
</application-policy>
```

## 2.4 Erster Start

Wenn man alles richtig gemacht hat, kann man nun JBoss aus der Kommandozeile heraus mit `run.bat` bzw. `run.sh` im Verzeichniss `JBOSS_HOME/bin` starten. Der Start kann je nach System etwas Zeit in Anspruch nehmen. Sobald in der Kommandozeile die Meldung erscheint, dass der Server hochgefahren wurde ('Started in <Zeitangabe>'), kann man die JBoss Startseite<sup>2</sup> aufrufen.

Der Login für die 'JBoss Administration Console' lautet: `admin/admin` (erster Login kann etwas länger dauern).

Zum Herunterfahren des Servers ruft man '`shutdown -S`' im Verzeichniss `JBOSS_HOME/bin` auf (dazu neue Kommandozeile öffnen).

---

<sup>2</sup><http://localhost:8080/>

# 3 JBPM

## 3.1 Download

Benötigt wird die JBPM Version 4.4<sup>1</sup>. Das heruntergeladene Zip-File wird in das Verzeichnis `yado/jbpm-4.4` entpackt.

## 3.2 Installation

ANT<sup>2</sup> wird benötigt, um die jBPM-Installationsskripte auszuführen. Das heruntergeladene Zip-File wird in ein Verzeichnis Deiner Wahl entpackt. Das Installationsverzeichnis muss nun als Wert für die Umgebungsvariable `ANT_HOME` gesetzt werden. Ausserdem muss das Bin-Verzeichnis zur Umgebungsvariable `PATH` hinzugefügt werden, damit Ant von der Kommandozeile aus aufrufbar ist.

Zum Testen, ob alles geklappt hat, einfach die Kommandozeile öffnen und den Befehl `'ant'` ausführen. Dann sollte man folgende Fehlermeldung erhalten:

```
Buildfile: build.xml does not exist!  
Build failed
```

### 3.2.1 HSQL-Datenbank

In der Kommandozeile ins Verzeichnis `yado/jbpm-4.4/install` wechseln. Dort wird `ant install.hsqldb.server` eingegeben.

Dieses Skript installiert die HSQLDB nach `yado\jbpm-4.4\hsqldb-server`. Nach der Installation starten wir die Datenbank über

```
yado/jbpm-4.4/hsqldb-server/start-hsqldb-server.bat.
```

In der Kommandozeile ins Verzeichnis `yado/jbpm-4.4/install` wechseln. Um die jbpm Schemas zu erzeugen gibt man folgendes ein:

```
ant -Ddatabase=hsqldb create.jbpm.schema
```

---

<sup>1</sup><http://sourceforge.net/projects/jbpm/files/jBPM%204/jbpm-4.4/jbpm-4.4.zip/download>

<sup>2</sup><http://artfiles.org/apache.org/ant/binaries/apache-ant-1.8.1-bin.zip>

Nachdem die Tabellen angelegt wurden, sollte man noch ein paar initiale Benutzerdaten in der Datenbank erzeugen, damit man sich später über die jBPM-Webanwendung einloggen kann:

```
ant -Ddatabase=hsqldb load.example.identities
```

### 3.3 Testen der Datenbank

Mit einem beliebigen DB-Tool Deiner Wahl eine Verbindung zur Datenbank erstellen.

Server-URL: jdbc:hsqldb:hsqldb://localhost:1701/

Login: sa

Passwort: (kein Passwort nötig)

Es müssten dann 18 jBPM-Tabellen vorhanden sein. In der Tabelle JBPM4\_ID\_USER sollten vier Benutzer angelegt worden sein.

### 3.4 jBPM im JBOSS

Der JBoss-Server sollte nicht laufen. Wir müssen zunächst folgende Datei editieren: yado/jbpm-4.4/install/build.xml. In der Zeile

```
<property name="jboss.parent.dir" value="${jbpm.home}" />
```

müssen wir den Wert so ändern, dass er auf das Verzeichnis verweist, in dem der JBoss liegt (yado\jboss). In der Kommandozeile nun ins Verzeichnis yado/jbpm-4.4/install wechseln. Dort folgendes aufrufen:

```
ant -Ddatabase=hsqldb install.jbpm.into.jboss
```

Danach zum Testen den JBoss starten (vorher HSQLDB starten, wenn die noch nicht läuft). Sobald der JBoss hochgefahren ist, rufen wir <http://localhost:8080/jbpm-console/> auf und können uns mit dem Login 'alex' und dem Passwort 'password' anmelden. Wenn wir dann eine leere Taskübersicht sehen, wurde jBPM erfolgreich im JBoss installiert.

# 4 YaDO

## 4.1 Grundlegendes

Die Anwendung besteht aus vier Dateien:

1. YadoService.jar - Die Serviceschicht
2. YadoProcess.jar - Die Prozessschicht
3. yado.war - Die Webschicht
4. yadows.war - Die Webservices

Zum Starten der Anwendung werden nur die Dateien 1-3 benötigt. Datei 4 ist optional. Beim deployen sollte die hier gezeigte Reihenfolge beachtet werden, da die Projekte aufeinander aufbauen.

## 4.2 Erster Start

Die Anwendung wird deployed indem man die Dateien in den Ordner `yado\jboss-5.1.0.GA\server\default\deploy` kopiert. Nachdem die Anwendung deployed wurde müssen nun die Rollen und der erste Administrator erzeugt werden. Dazu ruft man die reset-Seite<sup>1</sup> auf. Danach kann man sich über die yado-Seite<sup>2</sup> einloggen. Der Standard-Administrator lautet:

Name: administrator

Passwort: <ohne Passwort>

Nach dem ersten Start sollte ein neuer Administrator erzeugt werden und der Standard-Administrator deaktiviert werden. Da Benutzer und Getränke erst bei neuen Abrechnungszeiträumen übernommen werden, sollten diese erzeugt werden bevor man den ersten Abrechnungszeitraum startet.

---

<sup>1</sup><http://localhost:8080/yado/reset>

<sup>2</sup><http://localhost:8080/yado>

## 4.3 Webservices

Wenn die Webservices deployed wurden, dann sind die einzelnen Webservices unter folgenden URLs zu finden:

- <http://localhost:8080/yadows/accountingperiodservice>
- <http://localhost:8080/yadows/imageservice>
- <http://localhost:8080/yadows/transactionservice>
- <http://localhost:8080/yadows/drinkservice>
- <http://localhost:8080/yadows/inventorycontrolservice>
- <http://localhost:8080/yadows/orderservice>
- <http://localhost:8080/yadows/paymentrequestservice>
- <http://localhost:8080/yadows/statisticsservice>
- <http://localhost:8080/yadows/tallysheetservice>
- <http://localhost:8080/yadows/userservice>
- <http://localhost:8080/yadows/accountingperiodmgr>
- <http://localhost:8080/yadows/onlinepurchasemgr>
- <http://localhost:8080/yadows/ordermgr>

# Literaturverzeichnis

- [Agile Alliance 2001] AGILE ALLIANCE: *The Agile Manifesto*. <http://agilemanifesto.org/>. Version: 2001
- [Aier u. Schönherr 2006] AIER, Stephan ; SCHÖNHERR, Marten: *Enterprise application integration: Serviceorientierung und nachhaltige Architekturen ; [2. EAI-Expertentag, November 2003]*. 2. durchges. Berlin : GITO-Verl., 2006. – ISBN 978-3936771749
- [Baeyens u. Faura 2007] BAEYENS, Tom ; FAURA, Miguel V.: *The Process Virtual Machine*. <http://onjava.com/pub/a/onjava/2007/05/07/the-process-virtual-machine.html>. Version: 2007
- [Barrez 2009] BARREZ, Joram: *jBPM4 Hello World*. <http://www.jorambarrez.be/blog/2009/07/01/jbpm4-hello-world/>. Version: Juli 2009
- [Becker u. a. 2009] BECKER, Jörg ; MATHAS, Christoph ; WINKELMANN, Axel: *Geschäftsprozessmanagement*. Springer, 2009. – ISBN 978-3-540-85153-0
- [Biswas u. Ort 2006] BISWAS, Rahul ; ORT, Ed: *The Java Persistence API - A Simpler Programming Model for Entity Persistence*. <http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html>. Version: 2006
- [BPMI 2004] BPMI: *Business Process Model and Notation (BPMN)*. [http://www.bpmn.org/Documents/BPMN\\_V1-0\\_May\\_3\\_2004.pdf](http://www.bpmn.org/Documents/BPMN_V1-0_May_3_2004.pdf). Version: Mai 2004
- [Christensen u. a. 2001] CHRISTENSEN, Erik ; CURBERA, Francisco ; MEREDITH, Greg ; WEERAWARANA, Sanjiva: *Web Services Description Language (WSDL) 1.1*. <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>. Version: 2001
- [Cohn 2004] COHN, Mike: *User stories applied: For agile software development*. Boston, Mass. : Addison-Wesley, 2004 (Addison-Wesley signature series). <http://www.gbv.de/dms/bowker/toc/9780321205681.pdf>. – ISBN 0321205685
- [Cumberlidge 2008] CUMBERLIDGE, Matt: *JBoss jBPM Concepts and jBPM Process Definition Language (jPDL)*. <http://www.packtpub.com/article/jboss-jbpm-concepts-jpdl-jbpm-process-definition-language>. Version: Juni 2008
- [DeMichiel u. Keith 2006a] DEMICHEL, Linda ; KEITH, Michael: *EJB 3.0 Simplified API / Sun Microsystems*. 2006. (JSR 220: Enterprise JavaBeans, Version 3.0). – Forschungsbericht



- [DeMichiel u. Keith 2006b] DEMICHEL, Linda ; KEITH, Michael: EJB Core Contracts and Requirements / Sun Microsystems. 2006. (JSR 220: Enterprise JavaBeans, Version 3.0). – Forschungsbericht
- [DeMichiel u. Keith 2006c] DEMICHEL, Linda ; KEITH, Michael: Java Persistence API / Sun Microsystems. 2006. (JSR 220: Enterprise JavaBeans, Version 3.0). – Forschungsbericht
- [eclipse.org ] ECLIPSE.ORG: *help*. <http://help.eclipse.org>
- [ETH Zürich ] ETH ZÜRICH: *An Open Source Software Development And Collaboration Platform*. <http://www.origo.ethz.ch/>
- [Gadatsch 2010] GADATSCH, Andreas: *Grundkurs Geschäftsprozess-Management*. Vieweg+Teubner, 2010. – ISBN 978-3-8348-0762-5
- [Gloger 2008] GLOGER, Boris: *Scrum: Produkte zuverlässig und schnell entwickeln*. München : Hanser, 2008 <http://www.gbv.de/dms/ilmenau/toc/55238500X.PDF>. – ISBN 9783446414952
- [IBM ] IBM: *IBM WebSphere Process Server*. <http://www-01.ibm.com/software/integration/wps/>
- [JBoss Community a] JBOSS COMMUNITY: *Hibernate*. <http://www.hibernate.org>
- [JBoss Community b] JBOSS COMMUNITY: *jBPM*. <http://www.jboss.org/jbpm/>
- [JBoss Community c] JBOSS COMMUNITY: *jBPM User Guide*. [http://docs.jboss.com/jbpm/v4/userguide/html\\_single/](http://docs.jboss.com/jbpm/v4/userguide/html_single/)
- [JBoss Community d] JBOSS COMMUNITY: *jPDL*. [http://docs.jboss.org/jbpm/v4/userguide/html\\_single/#jpd1](http://docs.jboss.org/jbpm/v4/userguide/html_single/#jpd1)
- [JBoss Community e] JBOSS COMMUNITY: *The Process Virtual Machine*. [http://docs.jboss.org/jbpm/pvm/manual/html\\_single/](http://docs.jboss.org/jbpm/pvm/manual/html_single/)
- [JBoss Enterprise 2011] JBOSS ENTERPRISE: *JBoss Application Server*. <http://www.jboss.org/jbossas>. Version: 2011
- [Lafon 2002] LAFON, Yves: *W3C - Webservices*. <http://www.w3.org/2002/ws/>. Version: 2002
- [Lang ] LANG, Jean-Philippe: *The Flexible Project Management Web Application*. <http://www.redmine.org/>
- [Massol u. a. 2008] MASSOL, Vincent ; VAN ZYL, Jason ; PORTER, Brett ; CASEY, John ; SANCHEZ, Carlos: *Better Builds with Maven - The How-to Guide for Maven 2.0*. Exist Global, 2008 <http://www.maestrodev.com/better-build-maven>

- [Microsoft Corporation 2011a] MICROSOFT CORPORATION ; MICROSOFT CORPORATION (Hrsg.): *Microsoft Excel 2010*. <http://office.microsoft.com/de-de/excel/>. Version: 2011
- [Microsoft Corporation 2011b] MICROSOFT CORPORATION ; MICROSOFT CORPORATION (Hrsg.): *Microsoft Office*. <http://office.microsoft.com/de-de/>. Version: 2011
- [Newcomer 2002] NEWCOMER, Eric: *Understanding Web services: XML, WSDL, SOAP, and UDDI*. 2002
- [OMG 2006] OMG: *Business Process Model Notation (BPMN) Specification*. [http://www.bpmn.org/Documents/OMG\\_Final\\_Adopted\\_BPMN\\_1.0\\_Spec\\_06-02-01.pdf](http://www.bpmn.org/Documents/OMG_Final_Adopted_BPMN_1.0_Spec_06-02-01.pdf). Version: Februar 2006
- [OMG 2008] OMG: *Business Process Model and Notation, V1.1*. <http://www.omg.org/spec/BPMN/1.1/PDF>. Version: Januar 2008
- [OMG 2009] OMG: *Business Process Model and Notation, (BPMN), Version 1.2*. <http://www.omg.org/spec/BPMN/1.2/PDF>. Version: Januar 2009
- [OMG 2010] OMG: *Business Process Model and Notation (BPMN), Version 2.0*. <http://www.omg.org/spec/BPMN/2.0/PDF>. Version: Juni 2010
- [OpenOffice.org ] OPENOFFICE.ORG ; OPENOFFICE.ORG (Hrsg.): *OpenOffice.org - die freie Bürosoftware*. <http://de.openoffice.org/>
- [Oracle a] ORACLE: *JAAS Reference Guide*. <http://download.oracle.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html>
- [Oracle b] ORACLE: *JDBC Basics*. <http://download.oracle.com/javase/tutorial/jdbc/basics/index.html>
- [Ora 2010] ORACLE (Hrsg.): *The Java EE 5 Tutorial*. 821–1743–10. Oracle, Juni 2010. <http://download.oracle.com/javase/5/tutorial/doc/javaetutorial5.pdf>
- [Oracle 2011] ORACLE: *JAX-WS Reference Implementation*. <http://jax-ws.java.net>. Version: 2011
- [O'Reilly Media, Inc. ] O'REILLY MEDIA, INC.: *The Process Virtual Machine*. <http://onjava.com/pub/a/onjava/2007/05/07/the-process-virtual-machine.html>
- [Panda 2004] PANDA, Debu: *Simplifying EJB Development with EJB 3.0*. [http://www.oracle.com/technology/tech/java/newsletter/articles/simplifying\\_ejb3.html](http://www.oracle.com/technology/tech/java/newsletter/articles/simplifying_ejb3.html). Version: Oktober 2004

- [Pichler 2008] PICHLER, Roman: *Scrum - agiles Projektmanagement erfolgreich einsetzen*. 1. Aufl., korrigierter Nachdr. Heidelberg : dpunkt-Verl., 2008 [http://deposit.d-nb.de/cgi-bin/dokserv?id=2993206&prov=M&dok\\_var=1&dok\\_ext=htm](http://deposit.d-nb.de/cgi-bin/dokserv?id=2993206&prov=M&dok_var=1&dok_ext=htm). – ISBN 9783898644785
- [Pivotal Labs, Inc. ] PIVOTAL LABS, INC.: *The Collaborative And Lightweight Project Management Tool*. <http://www.pivotaltracker.com/>
- [Rücker 2009] RÜCKER, Bernd: *Ein erster Blick auf jBPM 4*. <http://www.bpm-guide.de/2009/01/26/ein-erster-blick-auf-jbpm-4/>. Version: Januar 2009
- [Rodriguez 2006] RODRIGUEZ, Alex: *RESTful Web services: The basics*. <http://www.ibm.com/developerworks/webservices/library/ws-restful/>. Version: 2006
- [Royce 1987] ROYCE, W. W.: Managing the development of large software systems: concepts and techniques. In: *ICSE '87: Proceedings of the 9th international conference on Software Engineering*, IEEE Computer Society Press, 1987. – ISBN 0-89791-216-0, S. 328–338
- [Saks 2009] SAKS, Kenneth: *Enterprise JavaBeans™ (EJB™) 3.1 Technology Overview*. <http://developers.sun.com/learning/javaoneonline/j1sessn.jsp?sessn=TS-4605&yr=2009&track=javaee>. Version: 2009
- [Salatino 2009] SALATINO, Mauricio: *jBPM Developer Guide*. Packt Publishing, 2009. – ISBN 978-1-847195-68-5
- [Sangeetha u. Chinnici 2007] SANGEETHA, S. ; CHINNICI, Roberto: *Using Annotations on Java EE 5.0 Plattform*, Mai 2007. <http://today.java.net/article/2007/05/18/using-annotations-java-ee-50-platform>
- [Schwaber 2004] SCHWABER, Ken: *Agile project management with Scrum*. Redmond, Wash. : Microsoft Pr., 2004 (Microsoft Professional). <http://www.gbv.de/dms/bowker/toc/9780735619937.pdf>. – ISBN 9780735619937
- [Software-Initiative Deutschland e.V. 2011] SOFTWARE-INITIATIVE DEUTSCHLAND E.V. ; SOFTWARE-INITIATIVE DEUTSCHLAND E.V. (Hrsg.): *IT verliert Kontrolle über Geschäftsprozesse*. <http://www.softwareinitiative.de/news/aktuell/2011-05-19-ITKontrolle>. Version: 2011
- [Stearns 2001] STEARNS, Beth: *EJB 2.0 Container-Managed Persistence Example*. <http://java.sun.com/developer/technicalArticles/ebeans/EJB20CMP/>. Version: Juli 2001
- [Sun-Microsystems ] SUN-MICROSYSTEMS: *MVC nach Sun Microsystems*. <http://java.sun.com/blueprints/patterns/MVC-detailed.html>

- [The Apache Software Foundation a] THE APACHE SOFTWARE FOUNDATION: *POM reference*. <http://maven.apache.org/pom.html>
- [The Apache Software Foundation b] THE APACHE SOFTWARE FOUNDATION: *Tapestry*. <http://tapestry.apache.org>
- [The Apache Software Foundation c] THE APACHE SOFTWARE FOUNDATION: *Wicket*. <http://wicket.apache.org>
- [The hsql Development Group ] THE HSQL DEVELOPMENT GROUP: *HSQL Datenbank*. <http://hsqldb.org>
- [Tyagi 2006] TYAGI, Sameer: *W3C - Webservices*. <http://www.oracle.com/technetwork/articles/javase/index-137171.html>. Version: 2006
- [unknown 2010] UNKNOWN: *Setting Up Tools to Build Applications Using jBPM: Part 1*. <http://javabg.eu/2010/02/setting-up-tools-to-build-applications-using-jbpm-part-1/>.  
Version: 2010
- [w3c 2011] w3c: *World Wide Web Consortium*. <http://www.w3.org/Consortium/>.  
Version: 2011
- [Wikipedia-Versionsverwaltung ] WIKIPEDIA-VERSIONSVERWALTUNG: *Versionsverwaltung*. <http://de.wikipedia.org/wiki/Versionsverwaltung>
- [XML.org ] XML.ORG: *BPEL*. <http://bpel.xml.org/specifications>
- [Yu 2009] YU, Jeff: *Getting Started with jBPM 4.0 (Part II)*. <http://jeff.familyyu.net/2009/07/getting-started-with-jbpm-40-part-ii.html>.  
Version: Juli 2009