7want.com

Bachelorarbeit

Nikolaus Hammler
nobaq@sbox.tugraz.at

Graz, August 2008

Betreuer: Dipl.-Ing. Christian Safran
Begutachter: Univ.-Prof. Dipl.-Ing. Dr.techn. Frank Kappe

Institut für Informationssysteme und Computer Medien (IICM),
Technische Universität Graz,
Inffeldgasse 16c, A-8010 Graz

Zusammenfassung

Diese Seminararbeit bildet den theoretischen Teil meiner Bachelorarbeit. Ziel des Projektes war die Entwicklung von 7want.com, einer „Verschenken"-Community. Nach dem Vorbild von eBay sollen Verschenker auf der Onlineplattform Geschenke inserieren, für die sich Community-Mitglieder bewerben können.

Die Seminararbeit besteht aus zwei größeren Teilen. Das erste Kapitel befasst sich mit grundlegenden Analysen über virtuelle Communities. Fragen über die Definition einer virtuellen Community, über Klassifizierungsmöglichkeiten sowie über die Motive und Teilnahme an einer virtuellen Community werden erläutert und beantwortet. Auch über Erfolgsfaktoren von virtuellen Communities wurde recheriert und die wichtigsten Punkte festgehalten. Weiters bildet der Abschnitt über Geschäftsmodelle eine Zusammenfassung über mögliche Geschäftsmodelle im Internet. Es wurde versucht, diese Theorie bzw. die daraus gezogenen Schlüsse auf 7want umzulegen.

Der zweite Teil befasst sich mit der Beschreibung des Bachelorprojektes, also dem Projekt 7want. Dabei ist ein Kapitel den grundlegenden Überlegungen und Entwicklungsentscheidungen beim Design gewidmet. Ein weiteres Kapitel beschreibt einige Aspekte der Implementierung in PHP.

Inhaltsverzeichnis

1 Einleitung **3**
- 1.1 Motivation . . . 3
- 1.2 Aufgabenstellung . . . 3
  - 1.2.1 Was ist 7want? . . . 3
  - 1.2.2 Produkt-Geschenke und Zeit-Geschenke . . . 4
  - 1.2.3 Inserieren von Geschenken . . . 4
  - 1.2.4 Nachrichtensystem . . . 4
  - 1.2.5 Schenker der Woche und TOP-Geschenke . . . 4
  - 1.2.6 Das Geschenkarchiv . . . 4
  - 1.2.7 My7want . . . 5
  - 1.2.8 Bewertungen . . . 5
- 1.3 Übersicht der Kapitel . . . 5

2 Virtuelle Communities **6**
- 2.1 Was sind virtuelle Communities? . . . 6
- 2.2 Die Gründung einer virtuellen Community . . . 7
- 2.3 Klassifizierung virtueller Communities . . . 7
- 2.4 Motive . . . 8
- 2.5 Erfolgsfaktoren . . . 10
- 2.6 Geschäftsmodelle . . . 12

3 Design **14**
- 3.1 Wahl der Software . . . 14
  - 3.1.1 Programmiersprache . . . 14
  - 3.1.2 Datenbank . . . 14
  - 3.1.3 Framework . . . 15
  - 3.1.4 Projektverwaltung . . . 15
- 3.2 Anwendungskonzept - Das MVC-Konzept . . . 15
- 3.3 Zerteilung in Datenmodule . . . 16
- 3.4 Datenbank . . . 17
- 3.5 Die Modelle . . . 17
- 3.6 Die Action-Controller . . . 19

4 Implementierung **21**
- 4.1 Programmcodeorganisation . . . 21
- 4.2 Bootstrapping . . . 22
- 4.3 Fehlerbehandlung . . . 22
- 4.4 Modelle . . . 22

4.4.1 Listenmodelle 23
4.4.2 Einfache Modelle 23
4.5 Controller - Ableitungshierarchie 23
4.5.1 YWantController 24
4.5.2 HasCategoryModeController 24
4.5.3 HasMyController 24
4.6 Ausgabe 24
4.7 Mehrsprachigkeit 25
4.8 Formulare 27
4.9 E-Mails 28

5 Schlussfolgerung und Ausblick 30

Kapitel 1

Einleitung

1.1 Motivation

Das Internet in seiner heutigen Form, mit seiner hohen Verbreitung in allen Gesellschaftsschichten, ermöglicht es, virtuelle Communities aufzubauen, deren Möglichkeiten viel größer sind als in konventionellen „Offline"-Communities. Der augenscheinlichste Vorteil ist dabei, dass sich die Mitglieder treffen und austauschen können, ohne einen Fuß vor die Tür setzen zu müssen. Das bedeutet auch, dass die Mitglieder nicht örtlich eingeschränkt sind.

Seit der ersten virtuellen Community von 1985 hat sich viel geändert: Waren virtuelle Communities anfangs lediglich textbasierte Foren von Experten, so treffen sich heute Menschen aus allen Bevölkerungsschichten, um virtuelle Tagebücher zu führen, Urlaubsphotos online zu veröffentlichen oder auch einfach um sich miteinander auszutauschen.

Seit ich im Sommer 2007 erstmals den Auftrag erhalten habe, eine „Verschenken"-Plattform zu programmieren, haben sich die Anforderungen laufend geändert: Sollte 7want anfangs lediglich eine eBay[1]-artige Plattform zum Inserieren von Geschenken werden so ist nun eine Plattform für eine Community enstanden, bei der vor allem der soziale Aspekt im Vordergrund stehen soll: Kontakte zu knüpfen und interessante Menschen kennen zu lernen.

1.2 Aufgabenstellung

1.2.1 Was ist 7want?

Die Community „7want" ist an die Idee von eBay angelehnt. Der augenscheinlichste Unterschied dabei ist, dass nicht Dinge verkauft sondern verschenkt werden. Eigentliche Intention der Plattform ist es jedoch, nicht das Verschenken an sich zu ermöglichen, sondern vielmehr eine Community von Verschenkern zu entwickeln, bei der der soziale Aspekt im Mittelpunkt steht. So müssen sich die Interessenten innerhalb eines bestimmten Zeitraumes (*Bewerbungsphase*) für ein Geschenk mit möglichst kreativen Gründen bewerben. In einer anschließenden *Entscheidungsphase* hat der Verschenker Zeit, sein Geschenk an einen oder mehrere Bewerber zu verschenken und kann dies auch begründen.

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

1.2.2 Produkt-Geschenke und Zeit-Geschenke

Unterstützt wird die soziale Intention durch zwei Arten von Geschenken: So stellen einserseits *Produkt-Geschenke* materielle Dinge dar, während bei *Zeit-Geschenken* Zeit verschenkt wird, z.B. in Form eines gemeinsamen Kinobesuchs oder Ausüben einer Sportart.

Diese beiden Arten stellen zwei Bereiche der 7want Plattform dar. In beiden Bereichen werden die Geschenke unterschiedlich kategorisiert aufgelistet, wobei die Kategorieliste als Menü links erscheint.

1.2.3 Inserieren von Geschenken

Zum Inserieren eines Geschenkes wählt der Inserent eine Kategorie aus und gibt Details wie Titel, Untertitel und Geschenkbeschreibung an. Zusätzlich können dem Inserat Fotos hinzugefügt werden. Der Inserent kann dabei die Dauer der Bewerbungsphase festlegen. Die auf die Bewerbungsphase folgende Entscheidungsphase dauert drei Tage.

1.2.4 Nachrichtensystem

Eine weitere wesentliche Unterstützung der Kommunikation zwischen den Mitgliedern wird durch ein spezielles Nachrichtensystem erreicht. Die Kommunikation wird dabei immer zwischen zwei Mitgliedern dargestellt. In einem erweiterten Modus wird die Kommunikation in Form eines Kommunikationsstranges („Threads“) angezeigt.

In Kontakt treten darf vorerst nur der Verschenker mit seinen Bewerbern. Sobald dies geschehen ist, befinden sich beide Kontakte auf der jeweiligen *Friends*-Liste des anderen und können sich uneingeschränkt Nachrichten schicken. Dem Inserenten ist es somit möglich, auf kreative Bewerbungen zu antworten. Umgekehrt ist es ohne Aufforderung nicht möglich, dass sich ein Bewerber durch private Kommunikation Vorteile verschafft.

Für „Störefriede“ gibt es eine Liste *geblockter Kontakte*. Das geblockte Mitglied kann daraufhin nicht mehr mit dem Mitglied, von dem es geblockt wurde, kommunizieren und sich auch nicht mehr für dessen Geschenke bewerben.

1.2.5 Schenker der Woche und TOP-Geschenke

Sowohl für Produkt-Geschenke als auch Zeit-Geschenke werden jede Woche „Schenker der Woche“ bestimmt. Diese Mitglieder zeichnen sich durch rege Aktivität auf der Plattform aus. Weiters gibt es laufend unterschiedliche „TOP-Geschenke“. Das sind besonders interessante, witzige oder auch großzügige Geschenke.

Beide Listen werden vorerst vom Betreiber moderiert und bilden die Startseite der Plattform.

1.2.6 Das Geschenkarchiv

Ist die Entscheidungsphase für ein Geschenk beendet oder wurde es verschenkt, so wird das Inserat in das Archiv verschoben. In diesem Archiv können alle älteren Geschenke abgerufen werden.

1.2.7 My7want

Der Bereich *My7want* ist der Mitgliedsbereich für 7want-Mitglieder. In diesem Bereich werden übersichtlich alle Inserate und Bewerbungen sowie das persönliche Archiv aufgelistet.

Zusätzlich zum Zugriff des Nachrichtensystems und den Bewertungen werden hier auch persönliche Einstellungen getätigt. Es ist z.B. möglich, sein eigenes Foto hochzuladen oder einen Steckbrief zu erstellen. Diese Daten sind dann in den Benutzerdetails (nur für Mitglieder) sichtbar. Lediglich die Fotos der Schenker der Woche oder die Mitglieder mit den TOP-Geschenken werden auch Besuchern gezeigt.

1.2.8 Bewertungen

Dem eBay-Vorbild folgend wird jede erfolgreiche Transaktion vom Bewerber bewertet. Diese Bewertungen können in den Benutzerdetails des jeweiligen Mitgliedes eingesehen werden.

1.3 Übersicht der Kapitel

Dieser Abschnitt soll einen kurzen Überblick über die Struktur dieses Dokuments und den Inhalt der folgenden Kapitel geben.

2 Virtuelle Communities: Dieses Kapitel bietet eine theoretische Beschreibung zu virtuellen Communities allgemein und nimmt dabei auf die 7want Plattform Bezug. Es wird erklärt, was virtuelle Communities sind, wie sie entstehen und erfolgreich sind und wieso sich Menschen in virtuellen Communities engagieren. Der letzte Abschnitt beschreibt Geschäftsmodelle, die angewendet werden können.

3 Design: Dieses Kapitel beschreibt das technische Design für die praktische Umsetzung von 7want. Dabei wird das verwendete Konzept erklärt sowie die Schritte der Designphase zusammengefasst.

4 Implementierung: Dieses Kapitel beschreibt die technische Implementierung der Plattform. Aufgrund des Projektumfangs werden in diesem Kapitel nur die wichtigsten Aspekte behandelt.

5 Schlussfolgerung und Ausblick: Eine zusammenfassende Schlussfolgerung sowie ein Ausblick auf zukünftig mögliche Erweiterungen bilden das letzte Kapitel dieser Arbeit.

Kapitel 2

Virtuelle Communities

2.1 Was sind virtuelle Communities?

Eine sehr frühe Definition der virtuellen Communities stammt aus dem Jahre 1993 von Rheingold: „Virtuelle Communities sind soziale Vereinigungen, die sich im Netz bilden, wenn sich genügend Menschen finden, die sich mit ausreichenden menschlichen Gefühlen lange genug an Diskussionen beteiligen, um so persönliche Bekanntschaften im Cyberspace zu knüpfen“ (Rheingold, 1993). Diese Definition stammt aus einer Zeit, zu der das Internet noch hauptsächlich Bildungseinrichtungen untereinander vernetzte und sich so vor allem Wissenschafter untereinander austauschten (vgl. Laudon und Traver, 2006). Rheingold sieht dabei noch keinen wesentlichen Unterschied zu „nicht virtuellen Communities“: Mitglieder einer virtuellen Community treffen aufeinander um das zu tun, was andere in der echten Welt tun würden. Der augenscheinliche Unterschied ist: Es geschieht über Computerbildschirme (vgl. Hamman).

Eine neuere, aber dennoch ähnliche Definition stammt aus der Diplomarbeit von Petra Schubert: „Virtuelle Gemeinschaften beschreiben den Zusammenschluss von Individuen oder Organisationen, die gemeinsame Werte und Interessen miteinander teilen und die über längere Zeit mittels elektronischer Medien, orts- und (teilweise auch) zeitungebunden in einem gemeinsamen semantischen Raum (gemeinsame Begriffswelt) kommunizieren“. (Schubert, 2000)

Eine weitere Definition aus dem Bereich des Marketings fand der Unternehmensberater John Hagel. „Eine virtuelle Community ist eine Gemeinschaft, die sich selbst im Internet entwickelt, um gemeinsame Interessen zu verfolgen. Im Gegensatz zu bilateraler Kommunikation, z.B. zwischen dem Betreiber und den Mitgliedern, können virtuelle Communities ein großes Netz unter den Mitgliedern selbst spannen“ (vgl. Hagel und Singer, 1999, S. 88).

Obwohl diese Definitionen Geheimsamkeiten aufweisen gibt es Gründe, weshalb es keine eindeutige Definition für eine virtuelle Community gibt: Erstens sind virtuelle Communities ein vielseitiges Forschungsobjekt, das von vielen Standpunkten analysiert werden kann (z.B. Psychologie, Verwaltungswissenschaften oder Computerwissenschaften). Zweitens ist es bei Modewörtern in diesem Bereich schwierig, zwischen wissenschaftlichen Begriffen und Umgangssprache abzugrenzen (vgl. Leimeister u. a., 2004).

Zusammenfassend lassen sich daraus vor allem zwei Schlüsse ziehen: Einer-

seits unterscheidet sich eine virtuelle Community nicht wesentlich von „nicht virtuellen“ Communities (ausser der Tatsache, dass die Interaktion virtuell geschieht). Andererseits scheint es eine wesentliche Rolle zu spielen, dass sich Communities von selbst bilden. Für die 7want-Community bedeutet das, dass die Plattform lediglich die Rahmenbedingungen für die Bildung einer Community schaffen kann.

2.2 Die Gründung einer virtuellen Community

Eine Beobachtung, die sich mit der Definition einer virtuellen Community von Hagel deckt, findet Wenger (1998): Eine Community muss nicht erst gebildet werden, sie ist bereits in Form einer „Community of Practice“ vorhanden.

Das bedeutet, dass die Community für 7Want nicht erst gegründet werden muss, sondern dass die Plattform die Rahmenbedingungen schaffen muss, diese bereits vorhandene „Community von Verschenkern“ in die Onlineplattform einzugliedern.

Bei zu hoher Einmischung einer übergeordneten Organisation (z.B. dem Betreiber) können sich jedoch negative Auswirkungen auf das Bestehen der Community zeigen (vgl. Wenger, 1998). Das bedeutet, dass sich der Betreiber von 7want nicht zu stark in die Community einmischen soll.

2.3 Klassifizierung virtueller Communities

Eine virtuelle Community zu klassifizieren scheint keine leichte Aufgabe zu sein, da sich hier sehr unterschiedliche Aufteilungen finden.

So gibt es nach Michel Gensollen drei wesentliche Arten von virtuellen Communities, die sich vor allem mit produktorientierten Communities beschäftigen (vgl. Brousseau und Curien, 2007, S. 174 ff.):

- „experience-sharing communities“ sind Communities bei denen sich Menschen über verschiedene Produkte austauschen können, z.B. sich gegenseitig Hilfe oder Tipps geben.
- „epistemic communities“ tauschen Wissen untereinander aus und machen es möglich, Wissen zu verbreiten.
- „file-sharing communities“. Die Mitglieder tauschen untereinander kulturelle Güter oder auch Software aus.

Im Buch über E-Commerce von Kenneth Laudon und Carol Traver werden virtuelle Communities wie folgt klassifiziert (vgl. Laudon und Traver, 2006, S. 793):

- „General“: Hier werden bestimmte Themen diskutiert.
- „Practice“: Eine Gemeinschaft von Fachleuten, z.B. Musikinterpreten oder Programmierern.
- „Interest“: Eine Gemeinschaft bei der ein zentrales Interesse im Mittelpunkt steht, z.B. Spiele, Sport, Politik oder Lifestyle.

- „Affinity“: Eine Gemeinschaft, deren Mitglieder sich demographisch oder geographisch identifizieren, z.B. Frauen oder Afroamerikaner.
- „Sponsored“: In diese Kategorie fallen alle Communities, die von einer Firma oder nicht-kommerziellen Organisationen für einen bestimmten Zweck geschaffen wurden.

Paul Siegel unterscheidet folgende Typen einer Community (Siegel, 2003):

- gruppenthematisch: Sie bieten News, Informationen, Foren oder andere Sachen, die für Mitglieder interessant sind.
- talent-fördernd: Im Rahmen eines Wettbewerbs posten die Mitglieder z.B. das Drehbuch für einen Film und bewerten dabei andere Drehbücher. Haben Mitglieder eine bestimmte Anzahl an anderen bewertet, bekommen sie Punkte für ihr eigenes. Die Besten werden veröffentlicht oder für eine Produktion ausgewählt.
- Produktorientiert: Diese Communities entsprechen Webforen, die z.B. Firmen für ihre Produkte einrichten.
- Fachwissenorientiert: Menschen die sich für ein bestimmtes Thema interessieren, können sich zu dazu austauschen und ihr Fachwissen teilen.
- Virtuelle Firmen: Hier arbeiten Firmen mit anderen zusammen, um Fähigkeiten zu teilen und zu kombinieren.
- Industrievereinigungen: Eine Plattform, die nicht nur die Interessen der Firmen befriedigt, sondern auch Leute zusammenbringt, um gemeinsam über Branchenthemen zu diskutieren.
  z.B. `http://www.autobodyonline.com`
- Spezielle Projekte wie OpenSource Entwicklung, z.B. die Beteiligung vieler Programmierer an einem Programm. Eine solche Community fördert die Tendenz von Firmen, ihren Sourcecode offenzulegen (z.B. Netscape).

Weiters kann eine Community ganz einfach in verbraucherorientierte oder Unternehmen-zu-Unternehmen Communities unterschieden werden, wobei sich die Verbraucherorientierten weiter in geographische, demographische und themenspezifische Communities unterteilen lassen. (vgl. Herstatt und Tietz, 2005)

Ein Grund für diese unterschiedlichen und sich teilweise überlappenden Einteilungen liegt womöglich am bereits genannten Mangel einer klaren Definition. Ein weiterer Grund kann die Betrachtung aus unterschiedlichen Bereichen sein.

Einige Forscher meinen, dass sich eine virtuelle Community erst gar nicht eindeutig kategorisieren lässt und dass sich vorhandene Kategorien überlappen. (Leimeister u. a., 2004)

2.4 Motive

Wieso treten Menschen virtuellen Communities bei? Boetcher u. a. (2002) haben folgende Gründe für das Engagement in virtuellen Communities gefunden:

- Die Gemeinschaft: Die Mitglieder wollen neuen Leute kennenlernen oder Geschichten und Witze teilen. Dies ist vor allem mit klassische Foren oder Chaträumen möglich.
- Gemeinsam arbeiten
  - geschäftlich: Innerhalb oder zwischen Firmen ist es mit virtuellen Communities möglich, dass ein Team gemeinsam an Projekten arbeitet.
  - geographisch: Lokale Gruppen oder Vereine wie z.B. Fussballvereine, können virtuelle Communities, z.B. in Form von Foren abieten, um diese zum Informationsaustausch verwenden.
  - thematisch: Virtuelle Communities können für Menschen interessant sein, die sich für ähnliche Themen und Dinge interessieren, z.B. Politik oder Studium.
- Thematische Konversationen führen, wie es z.B. im Usenet geschieht.

Diese augenscheinlichen Gründe können aber wiederum als Art Klassifikation von virtuellen Communities verstanden werden.

Der wichtigste Grund für das Engagement in einer virtuellen Community ist jedoch Reputation. Den Mitgliedern ist es wichtig, dass sie z.B. mit ihren Beiträgen in Foren wiedererkannt werden und auch auffindbar sind (Herstatt und Tietz, 2005). Für eine 7want Community bedeutet das, dass die verschenkten Artikel der Mitglieder wieder auffindbar sein müssen. Diese Tatsache wird in Form des Geschenk-Archives berücksichtigt. Weiters wird bei jedem Mitglied angezeigt, wieviel er schon verschenkt und wieviele Geschenke er bereits erhalten hat.

Andere Aspekte wurden durch eine empirische Untersuchung gefunden. Die Untersuchung basiert auf zwei Befragungen aus dem Filesharing-Bereich, lassen sich aber auch auf virtuelle Communities übertragen. Es wurden fünf Kernmotive gefunden, die vor allem auch die Reputation der Mitglieder betonen (Clement u. a., 2005):

1. Positive Netzwerkeffekte: Die Mitglieder sind sich bewusst, dass sie mit ihrem Engagement zum Erfolg der Community beitragen. Vor allem am Anfang ist dieses Motiv von entscheidender Bedeutung. Das Ziel des Betreibers muss nun sein, die Erwartung in die positiven Netzwerkeffekte zu steigern.

2. Reziprozität: Die anbietenden Mitglieder müssen das Gefühl haben, dass ihr Engagement gewürdigt wird. Das „Lurkertum“-Phänomen soll dabei möglichst nicht auftreten. Dieses Phänomen tritt dann auf, wenn ein kleiner Teil der Community Inhalte produziert, aber ein großer Teil Inhalte konsumiert. Es müssen entweder Maßnahmen geschaffen werden, die eine überproporionale Anzahl an Konsumenten vermeiden oder Motivationen für Produzenten geschaffen werden.

3. Altruismus: Dieses Motiv bedeutet eine freiwillige Verfolgung des Gemeinwohls einer Community. Das beste Beispiel für dieses Motiv findet sich in der OpenSource Community, bei der die einzige Kompensation der Programmierleistung in der Anerkennung innerhalb der Community besteht.

Unabhängig davon sollten Mechanismen installiert werden, die das Engagement der Mitglieder honorieren.

4. Anerkennung: Ein weiteres zentrales Motiv ist die bereits angesprochene Reputation. Diese kann gefördert werden, indem z.B. Rankings von Mitgliedern erstellt werden. Im Fall von 7want wird dieses Motiv vor allem durch die Schenker der Woche gefördert.

5. Kosten für Produzenten: Die Kosten für das Bereitstellen von Inhalten müssen so niedrig wie möglich sein und dürfen auf keinen Fall signifikant steigen. Das bedeutet für 7want, dass die Hürde zum Einstellen von Inseraten möglichst niedrig sein soll.

2.5 Erfolgsfaktoren

Communities sind im Internet sehr erfolgreich, das zeigen unzählige erfolgreiche Websites, die oft ihre Erfinder zu Millionären machten (z.B. Mark Zuckerberg mit der Studierendenplattform Facebook[1]). Ein Hauptgrund für den Erfolg von virtuellen Communities mag sein, dass sich die Mitglieder nicht in örtlicher Nähe befinden müssen sondern über den gesamten Erdball verstreut sein können (Richter, 2001).

Zu beachten ist, dass virtuelle Communities „Kritische-Masse-Systeme“ sind. Das bedeutet, dass sie dem Gesetz wachsender Erträge folgen. Eine Community mit wenigen Mitgliedern bringt nur einen geringen Nutzen. Ab einem bestimmten Schwellwert wächst der Nutzen jedoch exponentiell an (vgl. Stapf, 2002). Das bedeutet, dass der Erfolg einer virtuellen Community an der Anzahl der Mitglieder gemessen werden kann.

Doch was macht speziell eine virtuelle Community erfolgreich und wie misst man deren Erfolg?

Paul Siegel definiert den Begriff der „echten Community“ („true community“). Eine echte Community ist dann gegeben, wenn sich jedes Mitglied einbringt und anderen Mitgliedern hilft. „Eine echte Community bedingt positive Beteiligung und Lernen“ (Siegel, 2003).

Es scheint jedoch eine schwierige Aufgabe zu sein, den Erfolg einer Community genau zu definieren, da keine Definitionen gefunden werden konnten. Viel leichter scheint es zu sein, Gründe für den Erfolg von Communities zu finden, da diese einfach empirisch an Hand von Beispielen oder Umfragen gezeigt werden können.

Intuitiv möge man annehmen, dass vor allem ein breites Angebot und eine große Firma den Erfolg einer Community garantieren können. Aber dies ist nicht zwangsläufig der Fall: Apple gründete Mitte der Neunziger die Plattform „E-World“ mit einer großen Anzahl an Angeboten für Mitglieder, z.B. technischen Support direkt durch Apple. Obwohl Betatester der Plattform eine großartige Zukunft vorraussagten, musste diese bereits zwei Jahre später wieder geschlossen werden. An diesem Beispiel lässt sich einfach erkennen, dass eine Community nicht erfolgreich sein kann, wenn sie keine Akzeptanz unter den Besuchern bekommt. (Richter, 2001)

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

Als Beispiel für eine sehr erfolgreiche Community sei das Projekt „The WELL“[2] genannt. Die Community wurde 1985 gegründet und ist daher eine sehr alte Community. Auch Rheingold bezog seine Definition einer virtuellen Community auf „The WELL“. Für die Mitgliedschaft wird eine Mitgliedsgebühr eingehoben. Dies ist ein Beispiel dafür, dass „freie Mitgliedschaft“ für eine erfolgreiche Community nicht zwangsläufig notwendig ist. (Richter, 2001)

Doch was muss ein Projekt wie 7want speziell beachten um möglichst erfolgreich zu werden? Eine ganze Reihe konkreter Erfolgsfaktoren für virtuelle Communities haben Herstatt und Tietz (2005) aufgestellt:

1. Benutzer müssen sich registrieren können, vor allem damit sie ihr eigenes Profil schaffen können. Ein weiterer Vorteil ist dabei die Sammlung von Informationen zu Mitgliedern, die dann zielgerecht eingesetzt werden können

2. Vertrauen: Bevor der Benutzer seine persönlichen Daten preisgibt, will er sich sicher sein, dass seine Daten vertraulich behandelt werden. Dieser Punkt ist anfangs sehr schwer zu realisieren und lässt sich nur über einen längeren Zeitraum aufbauen

3. Regeln und Restriktionen: Verhaltensregeln in einer Community sind wichtig, denn ohne Regeln entsteht keine geordnete Kommunikation. Und dies animiert Mitglieder nicht, sich einzubringen. Auf der anderen Seite soll es aber möglichst keine Restriktionen, schon gar nicht in Form einer Zensur geben: Es darf nicht der Eindruck der Überwachung bestehen. Speziell für die Mitglieder der Woche und die TOP-Geschenke bedeutet dies, dass die Auswahl sehr neutral und dezent durchgeführt werden muss.

4. Commitment: Die Community soll nicht „nebenbei“ betrieben werden, sondern vom Betreiber gewartet werden. Dadurch soll den Mitgliedern das Gefühl von Wertschätzung entgegengebracht werden

5. Moderation: Für Streitigkeiten unter Mitgliedern soll es eine Moderation geben. Dadurch können Diskussionen auch in die richtige Richung gelenkt werden. Anfangs soll die Moderation vom Betreiber gelöst werden, kann später aber an erfahrene Mitglieder abgegeben werden

6. Einbeziehung der Mitglieder: Die Mitglieder sollen aktiv in Entscheidungen der Community eingebunden werden. Das stärkt das Gemeinschaftsgefühl innerhalb der Community und hebt das Niveau

7. In regelmäßigen Abständen soll Feedback von den Mitgliedern eingehoben werden, z.B. durch direkte Kontaktaufnahme, Fragebögen oder auch persönliche Treffen. Geschieht dies nicht, besteht die Gefahr, dass eine Community veraltet

Vor allem scheint Vertrauen ein wichtiger Faktor für den Erfolg einer Community zu sein, genauer: das Vertrauen in den Betreiber. Stapf (2002) meint dazu sogar: „Notwendige Voraussetzung für den Erfolg einer virtuellen Community ist die Vertrauenswürdigkeit des Betreibers.“.

[2] `http://www.thewell.com`

Ein weiterer wichtiger Faktor ist Kundenorientierung. Die virtuelle Community und alle ihre Geschäftsprozesse sollten dabei konsequent an die Bedürfnisse des Kunden ausgerichtet werden (Stapf, 2002).

2.6 Geschäftsmodelle

Für Anwendungen im Internet gibt es zahlreiche Geschäftsmodelle. Ein Geschäftsmodell beschreibt, mit welchen Methoden ein Unternehmen Gewinn machen kann. Im einfachsten Fall stellt ein Unternehmen Produkte her und verkauft diese. Sind die Einnahmen höher als die Produktionskosten, macht das Unternehmen Gewinn und das Geschäftsmodell war erfolgreich. (vgl. Rappa)

Für Anwendungen im Internet werden im Regelfall komplexere Geschäftsmodelle benötigt. Im groben unterscheidet Rappa dabei folgende Arten von Geschäftsmodellen:

- **Vermittlungsmodell**, bei denen das Unternehmen Anbieter und Kunden zusammenbringt und dafür eine Vermittlungsgebühr einhebt
- **Werbemodell**. Hier wird der Gewinn durch Einschaltung von Werbung erbracht und ist dann besonders erfolgreich, wenn die Werbung entweder viele Personen sehen oder sehr spezialisiert ist
- **Infomediärmodell**, bei dem eine Mittelsperson zwischen Anbieter und Konsument das Kaufverhalten sammelt und analysiert. Diese Daten sind sehr wertvoll, vor allem wenn sie in geordneter und analysierter Form vorliegen. Anhand solcher Analysen können gezielt Marketingkampagnen gestartet werden. Werbenetzwerke wie DoubleClick[3] wenden dieses Modell, indem anhand von Werbebannern ein Benutzerprofil erstellt wird
- **Handelsmodell**, bei dem das Unternehmen mit einem Onlineshop als Händler auftritt, wie z.B. das Onlinekaufhaus Amazon[4]
- **Herstellermodell**, bei dem ein Vertriebsweg online geschieht. Sehr einfach funktioniert das z.B. bei Lizenzen, da dafür keine Produkte verschickt werden müssen
- **Partnermodell**, bei denen prozentueller Anteil an einem Gewinn Anreize darstellen. Beispielsweise bietet Amazon an, Bücher über seine eigene Website zu verkaufen und bekommt dabei einen prozentuellen Anteil für jeden verkauften Artikel.
- **Communitymodell**, bei dem die in Abschnitt 2.4 genannten Motive wie Reputation oder Loyalität gegenüber der Community genutzt werden. Die Community erbringt eine Leistung, die der Betreiber verkaufen kann. Zum Beispiel vertreiben Linux Distributoren Software, die von der Community hergestellt wurden
- **Abonnementmodell**, bei dem Mitglieder in regelmäßigen Abständen für Leistungen bezahlen müssen

[3] `http://www.doubleclick.com`
[4] `http://www.amazon.com`

- **Nutzungmodell**, das ähnlich zum Abonnementmodell Mitgliedsgebühren einhebt, jedoch nicht auf regelmäßiger Basis sondern viel mehr auf Verlangen des Benutzers. Es wird nur das gezahlt was tatsächlich gebraucht wird. Beispielsweise werden Telefonate im Internet nach Zeit abgerechnet

Bei dieser Auflistung wird sofort klar, dass viele Geschäftsmodelle nur mit bestimmten Diensten genutzt werden können. Zum Beispiel kann das Herstellermodell nur angewendet werden, wenn der Betreiber tatsächlich Produkte herstellt. Geschäftsmodelle können jedoch auch kombiniert werden. Für 7want erscheint vor allem das Werbemodell, das Abonnementmodell oder eine Kombination daraus sinnvoll zu sein.

Vor allem für das Werbemodell werden entweder viele Benutzer benötigt oder aber eine spezialisierte Community, die bei 7want nicht vorhanden ist (vgl. Rappa).

Bei kommerziellen virtuellen Communities wird weiters unterschieden, ob das Kerngeschäft „online“ oder „offline“ stattfindet. Im ersten Fall ist die virtuelle Community ein zentraler Bestandteil der Geschäftstätigkeit, im zweiten Fall lediglich ein Teilaspekt. Findet das Kerngeschäft „online“ statt, so tritt das Unternehmen meist als Community-Betreiber auf. Der Erfolg hängt dabei direkt oder indirekt von der Anzahl der Mitglieder bzw. der Interaktion zwischen den Mitgliedern zusammen. Ziel dabei ist es, dass ein sogenannter Netzwerkeffekt entsteht. Ein Netzwerkeffekt ist ein Prozess, bei dem sich innerhalb kurzer Zeit durch positive Rückkopplung die Anzahl der Mitglieder exponentiell steigert. Dadurch wird in weiterer Folge erreicht, dass nachfolgende Wettbewerber die notwendige kritische Masse für den Erfolg einer Community nicht mehr erreichen können. (Clement u. a., 2005).

7want ist eine kommerzielle virtuelle Community, bei der das Kerngeschäft online stattfindet. Daher muss vor allem sichergestellt werden, dass der besagte Netzwerkeffekt eintritt. Dadurch kann auch erreicht werden, dass nachgeahmte Plattformen möglichst nicht erfolgreich werden.

Hagel und Armstrong haben für die Mitgliederentwicklung einer virtuellen Community ein vierstufiges Modell aufgestellt (Richter, 2001):

1. Locke Mitglieder an: Durch gezieltes Marketing und attraktiven Inhalten sollen Mitglieder angelockt werden. Es sollen keinesfalls Mitgliedsgebühren eingehoben werden

2. Fördere die Beteiligung: Die Mitglieder sollen dazu animiert werden, selbst Inhalte zu produzieren

3. Baue Loyalität auf: Zwischen den Community-Mitgliedern sollen Beziehungen aufgebaut werden, aber auch zwischen dem Betreiber und den Mitgliedern

4. Fahre Profit ein: Erst wenn sich die Community etabliert hat, soll spezielle Werbung geschaltet werden oder Gebühren eingehoben werden

Vor allem aufgrund des ersten Punktes sollen am Anfang keine Mitgliedsgebühren eingehoben werden. Das Abonnementmodell kann jedoch im vierten Schritt angewandt werden, wenn sich die Community etabliert hat.

Kapitel 3

Design

3.1 Wahl der Software

Vor der Erstellung des Designs schien es sinnvoll, zuerst die zu verwendende Software festzulegen. Vor allem war dabei von Interesse, welche Programmiersprache Verwendung finden soll und welches Framework eingesetzt werden soll - oder ob überhaupt ein Framework verwendet werden soll.

3.1.1 Programmiersprache

Die Wahl der Programmiersprache fiel auf PHP[1]. PHP ist eine etablierte, serverseitige OpenSource Scriptsprache, die vor allem in UNIX-Umgebungen eingesetzt wird. PHP ist OpenSource und ich hatte bereits einige Erfahrung in dieser Sprache, sodass diese, wenn möglich, bei 7want zur Anwendung kommen sollte.

3.1.2 Datenbank

Bei einer größeren Anwendung ist es vorteilhaft, wenn die Anwendungsdaten - im Falle von 7want die Benutzer oder Inserate - möglichst strukturiert gespeichert sind. Hier bietet sich ein relationales Datenbanksystem an, das über eine standardisierte Abfragesprache wie SQL auf die Daten zugreifen kann.

Die Wahl für 7want fiel auf mySQL[2], da mySQL bereits auf den verwendeten Servern zu Verfügung stand und ich bereits einige Erfahrungen mit mySQL hatte. mySQL bietet alle Funktionen, die für ein solches Projekt sinnvoll erscheinen. Von entscheidender Bedeutung sind dabei (vgl. Kofler, 2005, S. 5 ff.):

- SQL Kompatibilität
- Unterstützung von Unterabfragen (SubSELECTs)
- UTF-8
- Integrierte Volltextsuche
- Geschindigkeit: mySQL gilt als sehr schnelle Datenbank

[1] `http://www.php.net`
[2] `http://www.mysql.com`

3.1.3 Framework

Die Suche nach einem passenden Framework zur Unterstützung war nicht einfach, da es sehr viele Frameworks gibt und ein Einlesen in jedes Framework zu aufwändig ist. Mit PHP Frameworks hatte ich auch noch keine Erfahrungen. Ein Content Management System (CMS) wie Yoomla oder typo3 schied von vornherein aus, da es für andere Anwendungszwecke konzipiert wurde. Schlussendlich fiel die Wahl auf das Zend Framework[3]. Den Ausschlag gaben folgende Gründe:

- Das Framework baut auf PHP5 auf und ist daher komplett objektorientiert
- Vom Hersteller von PHP selbst
- Die Codequalität scheint durch strenge Richtlinien wie Codingstandards oder Unit Tests sehr hoch zu sein
- Das Framework bietet Unterstützung für das MVC-Konzept
- Die Datenbankabstraktion erschien sehr durchdacht zu sein. So wird einerseits das Table Gateway Pattern unterstützt, andererseits werden auch SELECT Statements als Objekte implementiert. Beim Table Gateway Pattern werden die Tabellen einer Datenbank als Objekte repräsentiert, was den Umgang mit Datenbanken erleichert, da vor allem für schreibende Operationen kein SQL Befehl erstellt werden muss.
- Es ist eine umfangreiche Dokumentation verfügbar

3.1.4 Projektverwaltung

Um eine Übersicht über den Entwicklungsverlauf zu haben oder alte Versionen wiederherstellen zu können wurde das Versionsverwaltungssystem Subversion[4] verwendet. Dies erlaubt auch eine einfache Aufteilung in eine Entwicklungsplattform, eine Testplattform und eine offizielle Version, wie sie unter `http://www.7want.com` veröffentlicht wird.

3.2 Anwendungskonzept - Das MVC-Konzept

Das Model-View-Controller (MVC) Design Pattern trennt in einer Anwendung die Repräsentation und die Darstellung der Daten und deren Interaktion. Das Konzept der Trennung von Eingabe, Ausgabe und Verarbeitung von Daten gab es in der Softwarewelt schon lange. Mit dem MVC Design Pattern kann das Konzept auch auf GUI Anwendungen übertragen werden. Es wurde erstmalig für die Erstellung von graphischen Benutzeroberflächen in Smalltalk verwendet. (vgl. Kaufman, 2001)

Beim MVC-Konzept entspricht die Ausgabe dem View, die Repräsentation der Daten dem Modell und der Interaktion oder Verarbeitung dem Controller.

Heute hat sich das MVC-Konzept vor allem auch im Bereich der Webanwendungen durchgesetzt. Vor allem für mittlere bis große Projekte ist es zu einem

[3] `http://framework.zend.com`
[4] `http://subversion.tigris.net`

unverzichtbarem Hilfmittel geworden. Es gibt aber keine scharfe Definition des Konzepts. Es werden jedoch vor allem zwei Ausprägungen unterschieden (vgl. Singer, 2004):

1. seiten-zentrisch: Hier besteht jede einzelne Seite aus Controller, View und Modell
2. controller-zentrisch: Es gibt nur einen Controller (*Front-Controller*) der nach Vorverarbeitungsschritten an die passenden Views weiterleitet.

Das vom Zend Framework verwendete Konzept ist eine Kombination aus den genannten Ausprägungen: Ein Front-Controller (`Zend_Controller_Front`) fängt alle Anfragen auf und leitet an einen *Action Controller* weiter. Dieser wiederum bekommt vom Front-Controller alle Benutzereingaben übermittelt, holt sich die Daten vom Modell und übergibt diese dem View. Die Ausgabe des Views wird in einem Response-Objekt zwischengespeichert und am Schluss an den Webbrowser zurückgeschickt. Die View Komponente wird mit `Zend_View` implementiert.

Das MVC-Konzept scheint für dieses Projekt optimal geeignet zu sein.

3.3 Zerteilung in Datenmodule

Bevor man mit der Erstellung der Datenstruktur beginnt, ist es sinnvoll, zuerst die Daten in Module aufzuteilen. Diese Module stellen die „Hauptdaten“ der Anwendung dar, die dann durch Datentabellen in der Datenbank oder durch Modellobjekte in der Anwendung repräsentiert werden.

Hauptdaten zeichnen sich dadurch aus, dass es die eigentlichen Daten sind, die die Anwendung verarbeitet.

In der Applikation gibt es folgende Hauptdaten:

- *Users*: Repräsentiert einen angemeldeten Benutzer
- *Product*: Repräsentiert ein Inserat. Entweder ein Produktgeschenk oder ein Zeitgeschenk
- *Message*: Repräsentiert eine Nachricht zwischen Benutzern
- *Feedback*: Repräsentiert einen Bewertungseintrag
- *Bid*: Repräsentiert eine Bewerbung für ein Inserat

Daneben gibt es noch eine Reihe an Daten die nicht die Hauptdaten der Anwendung darstellen sondern vielmehr „Hilsdaten“ sind. Als solche sind sie statisch in der Datenbank eingetragen und sind keiner unmittelbaren Änderung durch die Anwendung unterworfen. Sie sind Teil der Projektanforderungen.

- *Category*: Repräsentiert eine Kategorie für ein Inserat
- *Country*: Repräsentiert ein Land, aus dem ein Benutzer kommt
- *Region*: Repräsentiert eine Region, z.B. das Bundesland, aus dem ein Benutzer kommt

- *Ort*: Repräsentiert einen Ort, aus dem ein Benutzer kommt, z.B. Graz
- *PosterValue*: Repräsentiert eine Frage für den Steckbrief eines Benutzers
- *Question*: Repräsentiert eine Frage für die „Passwort vergessen“ Funktion.

3.4 Datenbank

Der nächste Schritt besteht im Design der Datenbank. Hier wird entschieden, in welchen Tabellen die Daten gespeichert werden, welche Attribute diese haben sollen und vor allem wie sie verknüpft werden sollen. Für die Verknüpfung unterscheidet man grob zwei Möglichkeiten:

1. 1:1 Verknüpfung: Für jedes Element aus der ersten Tabelle gibt es genau ein Element aus der zweiten Tabelle. Beispiel: Jeder Benutzer ist genau mit einem Ort verknüpft
2. 1:n Verknüpfung: Für jedes Element aus der ersten Tabelle kann es mehrere Elemente aus der zweiten Tabelle geben. Beispiel: Ein Inserat kann mehreren Kategorien zugeordnet werden

Für die Implementierung der 1:n Verknüpfungen werden zusätzliche Verknüpfungstabellen benötigt.

Für 7want gilt folgende Konvention für die Benennung der Tabellen:

- Tabellen für Hauptdaten erhalten den Prefix `t_`
- Tabellen für die Hilfsdaten erhalten den Prefix `th_`
- 1:n Verknüpfungstabellen erhalten den Prefix `ts_`

Der Zusammenhang zwischen den Tabellen wird am besten durch ein Entity-Relationship-Modell (ERM) Diagramm veranschaulicht. Das ERM Diagramm von 7want ist in Abbildung 3.1 zu sehen. Sehr deutlich ist dabei zu erkennen, dass die meisten Daten mit den Benutzern und den Produkten verknüpft sind.

3.5 Die Modelle

Die Modelle kapseln die eigentlichen Daten. Es gibe jedoch keine genau definierten Richtlinien, wie ein Modell aufgebaut ist und welche Teile der Logik im Modell verbaut sein muss. Auch das Zend Framework gibt hier keine Vorgaben. Eine verbreitete Methode, um die Modelle in Zend Framework zu implementieren ist die Klassen von `Zend_Db` oder `Zend_Service` direkt zu verwenden. Beispielsweise würde ein Modell für die Benutzer eine Klasse darstellen, die einfach von `Zend_Db_Table` abgeleitet ist und dabei das Table Gateway Pattern umsetzt.

Daraus folgt, dass die Verknüpfung zu anderen Daten im Action-Controller geschehen muss.

In 7want sind die Modelle einfache Objekte, die gegebenenfalls Zend_Db Klassen kapseln. Beim Design der Modelle half folgende Beobachtung: Treten mehrere Daten des gleichen Typs (Listen) auf, so müssen diese im Regelfall

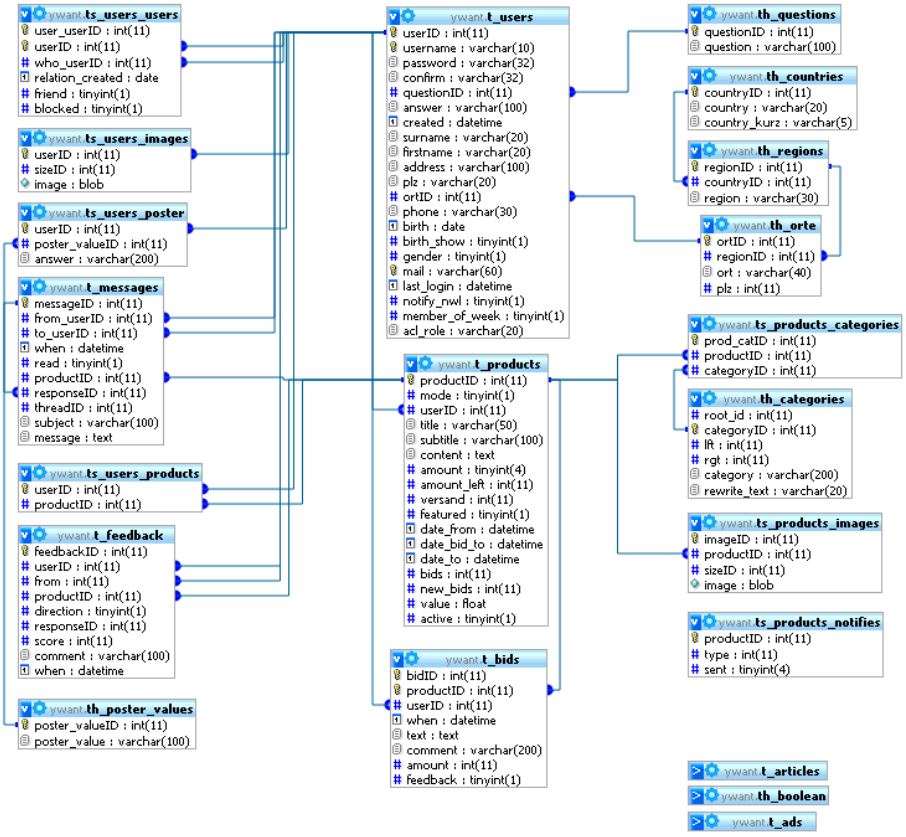

Abbildung 3.1: Entity-Relationship-Modell von 7want

nicht geändert werden, sondern lediglich der View-Komponente zur Ausgabe übergeben werden. Weiters werden zumeist in Listen die Daten verknüpfter Tabellen ebenfalls benötigt. Bei Daten die vereinzelt auftreten verhält es sich tendentiell umgekehrt. Deshalb soll es in 7want zwei unterschiedliche Typen von Modellen geben: Die einfachen Modellobjekte und die Listenmodelle.

Es soll zumindest für jeden Hauptdatentyp ein Modell existieren:

- Für *Users*: `UserModel` und `UserListModel`
- Für *Product*: `ProductModel` und `ProductListModel`
- Für *Message*: `MessageModel` und `MessageListModel`
- Für *Feedback*: `FeedbackModel` und `FeedbackListModel`
- Für *Bid*: `BidModel` und `BidListModel`

3.6 Die Action-Controller

Wird eine 7want Seite aufgerufen, so wird die URL durch den Front-Controller in ihre Bestandteile zerlegt und aufgrund dieser Informationen ein spezieller Action-Controller aufgerufen. Die Form der URLs in 7want ist /controllername/action/param1/wert1/.../param$N$/wert$N$. Beispielsweise würde die URL /product/show/productid/200 den Action-Controller `ProductController` instanzieren und darin die Methode `showAction` mit dem Parameter `productid` aufrufen.

Für jeden zusammengehörigen Codeblock soll ein Action-Controller existieren. Vorrangig sollen alle Methoden die z.B. ein bestimmtes Objekt benötigen, in einen Action-Controller zusammengefasst werden. Für 7want sollen folgende Action-Controller implementiert werden:

- `AboutController`: Beinhaltet alle statischen Seiten wie Impressum, AGB usw.
- `MyController`: Dieser Controller soll alle Methoden zur Änderung der eigenen Daten eines Benutzers zu Verfügung stellen, z.B. Passwort ändern, persönliche Daten ändern
- `MyUsers`: Soll die Listen „Meine Freunde“ und „Geblockte Kontakte“ im Bereich „My 7want“ implementieren
- `MywantController`: Der Hauptcontroller für den Bereich „My 7want“ soll die eigenen Inserate, Bewerbungen und vorgemerkten Inserate anzeigen und verwalten
- `ProductController`: Dient der Anzeige von einem Inserat und tätigt Aktionen die Bewerbungen
- `ProductlistController`: Soll der Anzeige von Inseratlisten dienen und die Ausgabe der Hauptseiten, der Kategorien und der Suche und des Bereichs „Geschenke kurz vor 7“ erstellen

- `UserController`: Beinhaltet alle Methoden die nicht von einem eingeloggten Benutzer abhängen: Benutzer erstellen, Benutzer anzeigen, Passwort vergessen

Zusätzlich gibt es zwei spezielle Action-Controller:

- `ErrorController`: Dieser Controller wird aufgerufen, wenn Exceptions im Controller nicht abgefangen werden. Er soll Fehlermeldungen anzeigen.
- `IndexController`: Dieser Controller wird aufgerufen, wenn kein Controller oder Action angegeben wurde. Er leitet einfach auf den Controller weiter, der als Startseite dienen soll

Kapitel 4

Implementierung

4.1 Programmcodeorganisation

Das Hauptverzeichnis des 7want Programmcodes besteht aus drei Verzeichnissen:

- `application`
- `public`
- `scripts`

Im Verzeichnis `application` befindet sich der PHP Programmcode für die Webapplikation, im Verzeichnis `public` alle Komponenten die öffentlich zugänglich sein sollen sowie die Bootstrap-Datei (siehe 4.2) und im Verzeichnis `scripts` befinden sich Shellscripts die entweder zur Wartung dienen oder für den Mailversand regelmäßig aufgerufen werden.

Das Verzeichnis `application` ist weiter aufgeteilt in:

- `config`: Beinhaltet Konfigurationsdateien sowie das Setup der Anwendung (z.B. Datenbankzugriff)
- `controllers`: Beinhaltet alle Action-Controller
- `languages`: Beinhaltet die übersetzten Dateien für die Mehrsprachigkeit
- `library`: Beinhaltet alle Klassen, die kein Controller, Action oder View sind. Das sind z.B. allgemeine Klassen, spezielle Basisklassen für Controller, Formulare oder Emails.
- `models`: In diesem Verzeichnis befinden sich alle Modelle sowie alle zugehörigen Objekte (Klassen für das Table Gateway Pattern und Exceptions)
- `views`: Beinhaltet die View-Komponenten: View-Scripts (das sind HTML Dateien, die nur mehr Verweise auf dynamische Daten beinhalten, also Templates), Mails und kleiner Helfer für die Ausgabe von Datei (View-Helper)

4.2 Bootstrapping

Im Gegensatz zu normalen PHP Anwendungen werden MVC Anwendungen zumeist so implementiert, dass alle Requests auf die Website an eine zentrale Stelle gehen, die den Front-Controller beinhaltet. Dieser Front-Controller ist in weiterer Folge für die Instanzierung der Action-Controller bzw. für den Aufruf der Actions verantwortlich. Diese Datei wird bootstrapfile genannt. Oft wird das Apachemodul mod_rewrite verwendet, um alle Requests auf die bootstrapfile umzuleiten. (vgl. McArthur, 2008, S. 203)

Im Falle von 7want wird folgende Regel verwendet, um alle Requests auf die index.php Datei umzuleiten. Diese ist die einzige PHP Datei die sich im public-Verzeichnis befindet:

```
RewriteEngine on
RewriteRule !public\\/ public/index.php
RewriteRule [a-z]{1,3}/public/(.*) public/\$1
```

Die bootstrap-Datei ist weiters für die Initialisierung der Anwendung verantwortlich, z.B. das Laden von Dateien, der Konfiguration Parsen der Request URL und schließlich der Instanzierung der Action-Controller. (vgl. McArthur, 2008, S. 203)

Dazu wird die Datei `boot_web.php` eingebunden, die sich im `config` Unterzeichnis befindet.

4.3 Fehlerbehandlung

Die Applikation ist so gestaltet, dass Fehler möglichst nur an einer Stelle abgefangen werden müssen. Dies wird durch die Verwendung von Exceptions realisiert. Bereits die Modelle werfen im Fehlerfall Exceptions, z.B. wenn ein Benutzer oder ein Inserat nicht gefunden werden kann. Soll die Exception bereits im Action-Controller bearbeitet werden, so wird sie einfach in der Action abgefangen. Alle anderen Exceptions reicht das Framework automatisch an den Action-Controller `ErrorController` weiter. In diesem Controller befindet sich nun für jede Exception eine Methode, die eine passende Fehlerseite ausgibt.

Auf diese Art und Weise kann die Anwendung durchgängig übersichtlich programmiert werden: Es wird einfach so programmiert ale würden keine Fehler auftreten. Tritt doch ein Fehler auf, so erhält der Benutzer automatisch eine Fehlermeldung von einer zentralen Stelle.

4.4 Modelle

Die Modelle wurden wie bereits im Designteil beschrieben umgesetzt: Es wurden die Modelle in einfache Modellobjekte und Listenmodelle aufgetrennt. Die Konvention für 7want ist, dass jedes Modellobjekt immer gültig sein muss. Das bedeutet, dass die Daten bereits im Konstruktor eingelesen werden und im Fehlerfall eine Exception geworfen wird (z.B. `UserNotFoundException`). Es gibt keine „leeren“ Objekte: Soll ein neuer Eintrag erstellt werden, so hat dies über statische Methoden der Objekte zu geschehen, z.B. mit `UserModel::create`.

4.4.1 Listenmodelle

Alle Listenmodelle sind von der Klasse `AbstractListModel` abgeleitet. Diese Klasse gibt einerseits eine API für alle Listenmodelle vor, andererseits bietet sie einfache Funktionen, um abgeleitete Modelle einfach zu verwenden: So braucht z.B. das `CountryListModel` in seinem Konstruktor lediglich `loadTable` aufrufen und das Modell enthält die gewünschten Daten.

Als Beispiel für ein komplexes Listenmodell sei `ProductListModel` genannt: Es kapselt alle Inseratelisten der Plattform. Die Art der Liste und deren Parameter wird dem Konstruktor übergeben. Die Daten des Modells werden über ein SELECT-Statement abgefragt. Dazu wird ein Objekt `ProductListSelector` verwendet das von `Zend_Db_Select` abgeleitet ist und einfache Methoden bereitstellt, die Abfrageergebnisse anzupassen. Beispielsweise lässt die Methode `limitToUser` nur diejenigen Inserate anzeigen, die der betreffende Benutzer eingestellt hat.

4.4.2 Einfache Modelle

Die einfachen Modelle verwenden in der Regel `Zend_Db_Table` und greifen direkt auf die betreffende Tabelle in der Datenbank zu. Jedoch hat sich gezeigt, dass diese Daten oft nicht ausreichen. Beispielsweise soll in einem Benutzerobjekt auch sein Wohnort dabei sein, ohne dafür eine extra Datenbankabfrage durchführen zu müssen. Deswegen sind diese Modelle wiederum in zwei Typen aufgeteilt, wobei die externe Schnittstelle durch das Interface `IModel` definiert ist:

- `AbstractTableModel`: Das Modell enthält nur die Daten der betreffenden Tabelle
- `AbstractTableModel`: Das Modell enthält auch Daten verknüpfter Tabellen. Dazu werden die Daten zwar mittels eines `Zend_Db_Select` Objektes ausgelesen, jedoch mit einem `Zend_Db_Table` Objekt geschrieben

Auf diese Art und Weise können alle Methoden nach dem gleichen Schema verwendet werden. Da die Modelle die Daten abstrahieren sollen, werden für keine Methoden der Modelle IDs übergeben (mit Ausnahme des Konstruktors), sondern immer Modellobjekte selbst. Beispielsweise erfordert die Methode `ProductModel::create` zum Erstellen eines neuen Inserats nicht die User ID des Inserenten sondern ein Objekt vom Typ `UserModel`. So ist automatisch sichergestellt, dass es den Benutzer für ein Inserat bereits gibt.

Weiters definieren alle Modelle Methoden um weitere Daten abfragen zu können. Zum Beispiel gibt die Methode `ProductModel::getUser` das betreffende `UserModel` für ein Inserat zurück.

4.5 Controller - Ableitungshierarchie

Die Action-Controller sind so aufgebaut, dass sie wenn möglich Aktionen mit ähnlichen Anforderungen zusammenfassen. Beispielsweise sind alle Aktionen die zwingend ein Inserat (und damit das Modell `ProductModel`) benötigen im

`ProductController` zu finden. Das hat den Vorteil, dass die Existenz des Inserates nicht in jeder Aktion abgefragt werden muss sondern nur bei der Instanzierung des Action-Controllers. Um solche Gemeinsamkeiten für mehrere Action-Controller zu definieren, gibt es eine Ableitungshierarchie. Die Basisklassen befinden sich im `library` Verzeichnis.

4.5.1 YWantController

Der `YWantController` ist die Basisklasse für alle Action-Controller in 7want. Er beinhaltet z.B. die Initialisierung der Ajax Komponenten oder weist dem View automatisch Variablen zu, die auf allen Seiten benötigt werden. Weiters definiert er Funktionen wie `needLogin` oder `redirect`, die Action-Controller verwenden können.

4.5.2 HasCategoryModeController

Dieser Controller ist die Basisklasse für alle Action-Controller, die zwingend eine Kategorie oder einen Mode (Produkt-Geschenk oder Zeit-Geschenk) vorraussetzen. Die betreffenden Modelle werden bereits bei der Instanzierung erstellt und im Fehlerfall wird eine Exception geworfen.

4.5.3 HasMyController

Dieser Controller ist die Basisklasse für alle Bereiche die einen Login vorraussetzen und damit auch einen gültigen Benutzer. Bereits bei der Instanzierung wird das Modell für den Benutzer erstellt und im Fehlerfall eine Exception geworfen. Alle Action-Controller, die von dieser Klasse erben, können nun sicher sein, dass der Benutzer eingeloggt ist, und auf den Benutzer mittels `$this->me` zugreifen.

4.6 Ausgabe

Für die Ausgabe der Inhalte wird zusätzlich zur View-Komponente noch eine Layout-Komponente verwendet, `Zend_Layout`. `Zend_Layout` implementiert das Two Step View Pattern:

Bei einer Webapplikation ist für gewöhnlich ein konsistentes Layout für jede Seite gewünscht. Beispielsweise besteht oft der Wunsch, globale Einstellungen an der Applikation sehr einfach vorzunehmen (z.B. dynamische Änderung des Headers). Der übliche Weg (Verwendung der reinen View Komponente) macht dies aber schwierig, weil die Information womöglich in jedem Viewscript vorkommt. Weiters besteht das Problem mit dem HTML Header: So ist es nicht möglich, dass ein Viewscript Veränderungen ausserhalb seines Bereichs vornimmt, zum Beispiel ein Script oder CSS Informationen in den Header schreibt.

Das Two Step View Pattern löst genau dieses Problem durch eine zweite Schicht: Im ersten Schritt werden wie gewohnt die Modelldaten verarbeitet, allerdings werden sie nicht direkt in den View eingefügt sondern in einer logischen Repräsentation zwischengespeichert. Erst der zweite Schritt fügt diese Komponenten zusammen in die Ausgabe ein. (vgl. Fowler)

Für die Ausgabe in 7want gibt es die Datei `default.phtml` im Unterverzeichnis `layouts`. Diese Datei enthält das Grundgerüst der Anwendung mit Header

und Footer. Der normale Inhalt, der von einem Action-Controller erzeugt wird, wird mittels

```
<?=$this->layout()->content?>
```

im Layout-Script eingefügt.

Jeder Action-Controller erzeugt seine Ausgabe, indem er Daten vom Modell abruft, und diese als normalen String oder Array dem View übergibt:

```
$this->view->assign('form', $form);
```

Nach Ablauf eines Controllers wird vom Framework selbst der View durch ein Plugin gerendert und die fertige Ausgabe im `content`-Objekt gespeichert welches dann am Schluss ins Layoutscript eingefügt wird. Vor allem andere Elemente im Header wie Seitentitel, CSS- oder Scriptangaben protitieren von dieser Technik. So werden sie im Layoutscript ausgegeben, dennoch kann jede Contentseite diese Objekte verändern. Dies soll am Beispiel des Seitentitels gezeigt werden. Im Layoutscript befindet sich die Ausgabe des Objekts:

```
<?
$this->headTitle()->prepend('7want.com');
$this->headTitle()->setSeparator(' - ');
echo $this->headTitle();
?>
```

Jedes View-Script kann nun einen weiteren Teil des Seitentitels anfügen, indem es es das `headTitle`-Objekt verändert:

```
<? $this->headTitle($this->_('Impressum')) ?>
```

Das Layout der HTML Seiten wurde nicht tabellarisch gestaltet sondern rein mit CSS. Cascading Stylesheets dienen einem Grundgedanken: der Trennung von Inhalt und Layout einer Website (Nefzger, 2007, S. 18). So kann das Layout der ganzen Anwendung durch die CSS Angaben gesteuert werden. Obwohl die reine Verwendung von CSS auch für das Layout anfangs etwas Einarbeitungszeit benötigt hat, hat sich die Verwendung gelohnt, denn: „Sinnvolle Argumente gegen CSS gibt es nicht“ (Nefzger, 2007, S. 18).

4.7 Mehrsprachigkeit

Eine Anforderung für 7want war eine einfache Übersetzung in andere Sprachen. Das wird zwar nicht unmittelbar gebraucht, jedoch sollten die Anwendung bereits so gestaltet werden, sodass eine einfache Übersetzung möglich ist.

Das Zend Framework bietet auch für dieses Anwendungsgebiet eine Lösung an: `Zend_Translate`. Die Vorteile von `Zend_Translate` sind (Zend):

- Es werden verschiedene Quellformate wie einfache Arrays oder CSV Dateien unterstützt, vor allem aber gettext[1], ein leistungsstarkes Paket zur Lokalisierung

[1] `http://www.gnu.org/software/gettext`

- Die API ist sehr einfach und einheitlich
- Die Sprache des Benutzers wird automatisch erkannt
- Die Quelldateien werden automatisch erkannt

7want verwendet das gettext Quellformat. Die Verwendung bringt zwei wesentliche Vorteile:

- Das Projekt kann ganz normal programmiert werden. Lediglich die zu übersetzenden Strings müssen markiert werden
- Die Übersetzung kann durch ein spezielles Programm erfolgen, das kein Wissen zur Anwendung erfordert. So kann die Übersetzung z.B. von einem eigenem Übersetzungsteam durchgeführt werden

Ein String wird mit der Methode `_` der Klasse `Zend_Translate` durchgeführt. Damit nicht überall das Translate-Objekt zu Verfügung stehen muss, wird in der `config.php` eine globale Funktion zur Übersetzung definiert:

```
function T_($str) {
        $t = Zend_Registry::get('translation');

        if($t && $t instanceof Zend_Translate)
        {
                return $t->_($str);
        }

        return $str;
}
```

Diese Funktion versucht das globale Translation-Objekt auszulesen. Falls es existiert, wird der übergebene String übersetzt. Falls nicht, wird der String einfach noch einmal zurückgegeben. Das erklärt bereits, wieso keine speziellen Vorkehrungen für die Übersetzung getroffen werden mussten: 7want wird einfach in Deutsch programmiert die Strings selbst werden als ID für die Übersetzung in andere Sprachen verwendet.

Um alle Strings für die Übersetzung auszulesen, wurde das Script `sync-po.sh` im `scripts`-Verzeichnis programmiert. Es durchsucht mithilfe der gettext Komponente *xgettext* alle Quelldateien nach Strings die mit `T_` ummantelt sind und schreibt das Ergebnis für jede Sprache in Dateien im languages Verzeichnis. Das Script ist dabei so gestaltet, dass es bereits vorhandene Übersetzungen nicht überschreibt und die vorhandenen Dateien mit neu gefundenen Informationen kombiniert.

Diese Sprachdateien können nun einer externen Übersetzungsgruppe zur Übersetzung vorgelegt werden. Ein weiterer Aufruf von `sync-po.sh` erzeugt die compilierte Dateien mit der Endung *mo*, auf die dann `Zend_Translate` für die Übersetzung zurückgreift.

Für die Auswahl der Sprache wurde ein Plugin für den Front-Controller verfasst. Dieses Plugin überprüft den ersten Paramter der URL und setzt die Sprache dementsprechend. Wird keine Sprache angegeben oder die angegebene Sprache nicht gefunden, so wird versucht die Sprache vom Browser auszulesen. Falls auch diese Sprache nicht existiert, wird Deutsch als Standardsprache gewählt.

4.8 Formulare

Die Behandlung von Formularen stellt eine Arbeit dar, bei der einige Aufgaben mehrfach erledigt werden müssen. Zum Beispiel:

- Zusammenstellung des Formulars und Generierung des HTML Codes
- Validierung der Eingaben
- Filterung der Eingaben
- Ausführung der Aktionen basierend auf den Eingaben

Alle diese Aufgaben kann in einfacher Form `Zend_Form` erledigen. Hierzu wird ein Formular als Objekt betrachtet und jedes Formularelement wird ebenfalls als Objekt repräsentiert. Für ein bestimmtes Formular wird nun am besten von `Zend_Form` abgeleitet und in der Initialisierungsmethode die Formularelemente hinzugefügt.

Mit verschiedenen Methoden werden die Eigenschaften der Formularelemente gesetzt, z.B. der Titel eines Elements oder der Name. Nun können noch Validatoren oder Filter hinzugefügt werden. Diese Elemente sind ebenfalls Objekte, die durch eine standardisierte Weise Eingaben überprüfen oder filtern.

Um auf einer Seite ein Formular zu verwenden, wird einfach ein Formularobjekt instanziert, z.B. `Form_UserCreate` zur Erstellung eines Benutzers. Für die Darstellung braucht nur das Formularobjekt dem View übergeben zu werden. Durch die Implementierung der `__toString` Methode wird es bei der Ausgabe automatisch samt seiner Elementen gerendert. Die Überprüfung geschieht einfach mit der `isValid`-Methode und die fertig gefilterten Werte können einfach mit `getValues` ausgelesen werden. Ein Formular kann dadurch sehr einfach verwendet werden, zum Beispiel:

```
$form = new Form_PwChange($this->me);

if($this->getRequest()->isPost())
{
        if($form->isValid($this->getRequest()->getPost()))
        {
                $values = $form->getValues();
                $this->me->setPassword($values['password1']);

                $this->_helper->getHelper('Redirector')
                        ->gotoAndExit('index', 'mywant');
        }
}

$this->view->assign('form', $form);
```

Im Fehlerfall wird das Formular einfach noch einmal angezeigt - samt seinen Fehlermeldungen. Waren die Eingaben erfolgreich, so werden die Werte einfach mit `getValues` ausgelesen, verarbeitet und schließlich auf eine andere Seite weitergeleitet.

4.9 E-Mails

In 7want müssen auch E-Mails versendet werden:

- Bestätigung der Accounterstellung
- Bestätigung einer neuen E-Mail Adresse
- Zusendung eines neuen Passwortes, falls dieses vergessen wurde
- E-Mail an den Bewerber, der ein Geschenk erhalten hat
- E-Mail an die Bewerber, die kein Geschenk erhalten haben
- E-Mail an den Inserenten, dass er keine Bewerbungen erhalten hat
- Erinnerung an den Inserenten, sich für einen Bewerber zu entscheiden
- Ein Statusmail mit Übersicht über neue Nachrichten und neue Bewerbungen

Alle diese E-Mails sollen auf die gleiche Art und Weise erstellt und verschickt werden. Erschwerend ist dabei, dass manche E-Mails durch eine direkte Aktion des Benutzers ausgelöst werden (z.B. Bestätigung der Accounterstellung). Manche E-Mails müssen jedoch in regelmäßigen Abständen verschickt werden.

Das Zend Framework bietet bereits ein Basismodul für das Versenden von E-Mails: `Zend_Mail`. Diese Komponente erlaubt das Versenden und die Erstellung von E-Mails, sowohl im Textformat als auch im HTML-Format oder kombiniert. Dazu wird ein `Zend_Mail` Objekt instanziert und Betreff, Absender, Empfänger und der Nachrichteninhalt übergeben.

Alle Mails die von der Applikation aus verschickt werden weisen bestimmte Gemeinsamkeiten auf, z.B. den gleichen Absender oder eine UTF8 Kodierung. Weiters soll der Inhalt der Nachricht genau wie eine Ausgabeseite der Anwendung aus einem View-Script generiert werden. Für die Zusammenfassung dieser Aufgaben wurde eine neue Klasse `YWantMail` von `Zend_Mail` abgeleitet. Dieser Klasse wird einfach der Name des View-Scripts übergeben, mit `setRecipient` ein `UserModel` übergeben und schließlich mit der Methode `send` das Mail verschickt.

Für jeden E-Mailtyp wird nun noch eine Klasse von `YWantMail` abgeleitet. Im jeweiligen Konstuktor werden dabei die Werte gesetzt, die ein bestimmtes Mail charakterisieren: Das View-Script, den Betreff und die Zuweisung der Daten ins Viewscript.

Die Mails, die regelmäßig verschickt werden benutzen die Klasse `CronMails`. Diese Klasse wird direkt in den cron-Scripts verwendet.

Eine spezielle Behandlung benötigt das Status-Mail: Es soll nur versendet werden, wenn entweder eine ungelesene Nachricht vorhanden ist oder eine neue Bewerbung für ein Inserat existiert. Dabei sollen aber alle Inserate aufgelistet werden. Für diese Aufgabe werden die Daten in der Klasse `CronMails` mit SELECT Statements ausgelesen und für jeden Benutzer mit passendem Kriterium ein `Mail_Status` Objekt erstellt. Diese Objekte werden in einem Array gespeichert. Gibt es ein neues Inserat, so wird die Methode `addProductNotify` aufgerufen. Mit der Methode `setNewMessagesCount` wird die Anzahl der ungelesenen Nachrichten festgesetzt. Zum Schluss wird von jedem E-Mail die Methode

`sendIfValid` aufgerufen. Diese überprüft, ob das E-Mail überhaupt verschickt werden muss und verschickt es gegebenenfalls.

Kapitel 5

Schlussfolgerung und Ausblick

Seit dem Beginn des Projektes im Sommer 2007 haben sich die Anforderungen an das System laufend geändert. Das zeigt wie wichtig es ist und war, die Plattform im Vorfeld genau zu planen. Die genaue Planung, aber auch Einhaltung des MVC-Konzepts und Einhaltung von Konventionen haben es ermöglicht, auf Änderungswünsche des Kunden möglichst schnell reagieren zu können und werden es hoffentlich auch ermöglichen, zukünftige Änderungswünsche rasch umsetzen zu können. Die Dynamik über den Entwicklungszeitraum war eine große Herausforderung und hat wertvolle Erfahrung gebracht.

Die Plattform selbst ist nun fertiggestellt und bietet die optimalen Rahmenbedigungen für die Entwicklung einer virtuellen Community. Die genannten Erfolgsfaktoren, soweit sie für 7want zutreffend sind, wurden weitestgehend befolgt und in der Plattform eingebaut.

Vor allem in der ersten Phase ist es sehr wichtig, dass die Plattform vom Betreiber aktiv gewartet wird und auf Wünsche aus der Community rasch eingegangen wird. Am besten sollen Mitarbeiter des Betreibers selbst am Geschehen in der Community teilnehmen. Frühe Anreize können vor allem durch Einstellen besonderer Geschenke des Betreibers erreicht werden. Die Implementierung eines Feedbackformulars würde als weiterer Erfolgsfaktor die Kommunikation zwischen Betreiber und Mitgliedern erleichtern.

Für die folgende kommerzielle Nutzung der Plattform sollte das vierstufige Modell von Hagel beachtet werden. Die Realisierung der ersten drei Stufen wird einige Zeit in Anspruch nehmen, ist jedoch zum Aufbau der Community sehr wichtig. Erst in der letzten Stufe sollte Werbung eingeblendet werden und über Mitgliedsbeiträge nachgedacht werden.

Technisch gibt es noch ein gewisses Entwicklungspotential; Der dafür notwendige Aufwand macht sich jedoch erst ab einer großen Anzahl an Besuchern und Mitgliedern bezahlt:

- **Caching**: Ein flexibles Cachingkonzept kann verhindern, dass bei jeder Aktion eine Datenbankabfrage ausgeführt werden muss. Das Zend Framework bietet dafür das Modul `Zend_Cache` an, mit dem sich ganze Objekte cachen lassen. Ganze Gruppen von gecachten Elementen können durch ein System mit Tags invalidiert werden. Der Cachingmechanismus ließe sich

direkt in die Datenmodelle einbauen, ohne dass weite Teile der Applikation geändert werden müssten.

- **Einbingung von GIS Daten**: Ähnlich wie bei eBay könnte die Entfernung zum Artikelstandpunkt anhand der Adressen der Mitglieder ermittelt und angezeigt werden.
- **Erweiterte Suche**: Die derzeitige Suche erlaubt lediglich das Auffinden von Geschenken nach einem Stichwort. Mit einer erweiterten Suche könnten Artikel nach anderen Kriterien gesucht werden, z.B. nach Kategorie, Ort des Geschenks, oder (mit Einbindung von GIS Daten) nach Entfernung. Weiters ist die Implementierung einer Fuzzy-Suche möglich, die auch ähnliche Worte findet.
- **Recommender System**: Ausgehend von vorhandenen Geschenken in der Datenbank könnten zu jedem Artikel ähnliche Geschenke angezeigt werden. Ein ähnliches System verwendet z.B. Amazon.

Literaturverzeichnis

[Boetcher u. a. 2002] Boetcher, Sue ; Duggan, Heather ; White, Nancy: *What is a Virtual Community and Why Would You Ever Need One?* `http://www.fullcirc.com/community/communitywhatwhy.htm`. 2002. – (21.08.2008)

[Brousseau und Curien 2007] Brousseau, Eric ; Curien, Nicolas: *Internet and Digital Economies*. Cambridge University Press, 2007

[Clement u. a. 2005] Clement, Michel ; Panten, Gregor ; Peters, Kay: Effiziente Kommunikation in Communities. In: *Thexis* 22 (2005), Nr. 3, S. 21 – 27

[Fowler] Fowler, Martin: *Two Step View*. `http://www.martinfowler.com/eaaCatalog/twoStepView.html`. – (26.08.2008)

[Hagel und Singer 1999] Hagel, John ; Singer, Marc: *Net Worth. Shaping Markets When Customers Make the Rules*. Mcgraw-Hill Professional, 1999

[Hamman] Hamman, Robin: *Introduction to Virtual Communities Research and Cybersociology Magazine Issue Two*. `http://www.cybersociology.com/files/2_1_hamman.html`. – (01.09.2008)

[Herstatt und Tietz 2005] Herstatt, Cornelius ; Tietz, Robert: Erfolgsfaktoren für den Aufbau und die Nutzung virtueller Communities. In: *Thexis* 22 (2005), Nr. 3, S. 47 – 51

[Kaufman 2001] Kaufman, Benjamin: *Das Model-View-Controller Modell*. `http://fara.cs.uni-potsdam.de/~kaufmann/tuts/mvc.pdf`. 2001. – (25.08.2008)

[Kofler 2005] Kofler, Michael: *The Definitive Guide to MySQL 5*. Third Edition. Apress, 2005

[Laudon und Traver 2006] Laudon, Kenneth C. ; Traver, Carol G.: *E-Commerce, business. technology. society.* Third edition. Pearson Prentice Hall, 2006

[Leimeister u. a. 2004] Leimeister, Jan M. (Hrsg.) ; Sidiras, Pascal (Hrsg.) ; Krcmar, Helmut (Hrsg.): *Success factors of virtual communities from the perspective of members and operators: An empirical study*. 2004

[McArthur 2008] McArthur, Kevin: *Pro PHP: Patterns, Frameworks, Testing and More*. Apress, 2008

[Nefzger 2007] Nefzger, Wolfgang: *Cascading Style Sheets für Profis*. Franzis Verlag GmbH, 2007

[Rappa] Rappa, Michael: *Business Models on the Web*. `http://digitalenterprise.org/models/models.html`. – (27.08.2008)

[Rheingold 1993] Rheingold, Howard: *The Virtual Community*. FIXXME, 1993

[Richter 2001] Richter, Wolfgang: *Virtual Communities und Customer Relationship Management*, Institut für Informationsverarbeitung und Computergestützte neue Medien (IICM), Technische Universität Graz, Diplomarbeit, 2001

[Schubert 2000] Schubert, Petra: *Virtuelle Transaktionsgemeinschaften im Electronic Commerce*. Josef Eul Verlag, 2000

[Siegel 2003] Siegel, Paul: *How Do We Develop Community?* `http://www.learningfountain.com/howbcomm.htm`. 2003. – (27.08.2008)

[Singer 2004] Singer, Leif: *Model-View-Controller*. `http://www.se.uni-hannover.de/documents/kurz-und-gut/ws2004-seminar-entwurf/mvc_lsinger.pdf`. 2004. – (26.08.2008)

[Stapf 2002] Stapf, Wolfang: Virtual Communities - aus prozessorientierter Sicht. In: *HMD* 224 (2002), S. 94 – 106

[Wenger 1998] Wenger, Etienne: *Communities of Practice: Learning, Meaning, and Identity*. First Paperback Edition. Cambridge University Press, 1998

[Zend] Zend: *Dokumentation zu Zend_Translate*. `http://framework.zend.com/manual/de/zend.translate.html`. – (05.09.2008)