Universität Koblenz-Landau
Institut für Wirtschafts- und Verwaltungsinformatik
Arbeitsgruppe Methoden und Modellbildung

„Simulations-Wiki, ESSA und SocSimNet“

Studienarbeit

vorgelegt von

Oliver Weichert

Matr.-Nr. 119820169

weichert@uni-koblenz.de

betreut durch

Prof. Dr. Klaus G. Troitzsch

07.12.2006

Inhaltsverzeichnis

1 Einführung....1
1.1 Gegenstand der Studienarbeit....1
1.2 Zielsetzung....1
2 Begriffe....2
2.1 Portal....2
2.2 Wiki....2
2.3 Zope....3
2.4 Plone....4
2.5 Content-Management-System (CMS)....5
3 Anforderungen....7
3.1 Gleichgebliebene Anforderungen....8
3.1.1 Benutzerfreundlichkeit....8
3.1.2 Möglichkeit zur Veröffentlichung von eigenen Artikeln....8
3.1.3 Möglichkeit zur Führung eines Veranstaltungskalenders....8
3.1.4 Verzeichnis aller Mitglieder....9
3.1.5 Möglichkeit zur Organisation von Links....9
3.1.6 Möglichkeit zum Anbieten von Downloads....9
3.2 Entfernte Anforderungen....10
3.2.1 Kommunikationsmöglichkeit....10
3.2.2 Möglichkeit zur Führung eines Journals....10
3.2.3 Hilfe-System (Kann-Anforderung)....10
3.2.4 Favoriten für das Journal (Kann-Anforderung)....11
3.3 Neue Anforderungen....11
3.3.1 Einbindung eines Wiki zur Verwaltung von Begriffen aus dem Bereich sozialwissenschaftliche Simulation....11
3.3.2 Integration einer Bibliographie zur Verwaltung von Literatur, die sich mit dem Thema sozialwissenschaftliche Simulation beschäftigt....11
3.3.3 Einbindung eines Kurskataloges (SocSimNet) zur Verwaltung von Kursangeboten, Dozenten und Veranstaltungsorten....12
3.3.4 Bindung an das Zielsystem Plone....12
3.4 Zusammenfassung....12
4 Implementierung des neuen Systems....13
4.1 Installation von Zope und Plone....13
4.2 Grundeinstellungen vornehmen....14
4.3 Anpassung der Templates....14
4.4 Wiki....16
4.5 Bibliographie....18
4.6 Newsletter....18
4.7 Umfrage-Tool....20
4.8 Workflows, Sicherheit und Benutzer....22
5 Implementierung des Kurskataloges (SocSimNet)....24
5.1 Einleitung....24
5.2 Archetypes....25
5.2.1 Einleitung....26
5.2.2 Schemata, Felder und Kontrollelemente....26
5.2.3 Ansichten (Views)....31
5.3 ArchGenXML....32
5.3.1 Einführung....32

5.3.2 Arbeitsweise und Beispiel....32
5.4 Implementierungsdetails....33
5.4.1 Struktur....34
5.4.2 Objekte....35
5.4.3 Views....39
5.4.4 Storage....41
5.4.5 Probleme, die während der Entwicklung auftraten....41
5.5 Zusammenfassung....42
6 Migration der Daten in das neue System....43
7 Ausblick....44

Abbildungsverzeichnis

Abbildung 1: Aufbau eines CMS [6].....5
Abbildung 2: Layout der bestehenden ESSA-Seite.....7
Abbildung 3: Plone Control Center.....14
Abbildung 4: ZMI (Zope Management Interface).....15
Abbildung 5: Neues Design der ESSA-Seite.....16
Abbildung 6: Startseite des Wikis.....17
Abbildung 7: Newsletter - Abonnenten Verwalten.....19
Abbildung 8: Poll - Abstimmung und Ergebnis im Portlet.....20
Abbildung 9: Poll - Umfrage im normalen Anzeigebereich.....21
Abbildung 10: Poll - Ergebnis im normalen Anzeigebereich.....22
Abbildung 11: Default-Workflow in Plone für Inhalte [McKay].....23
Abbildung 12: UML-Diagramm von SocSimNet.....25
Abbildung 13: Aufbau eines Schemas [PloneBuch].....27

Tabellenverzeichnis

Tabelle 1: Rechte im Default Workflow [McKay].....23
Tabelle 2: Verfügbare Felder [2].....27
Tabelle 3: Verfügbare Attribute der Felder [2].....28
Tabelle 4: Verfügbare Widgets [3].....29
Tabelle 5: Verfügbare Attribute für Widgets [3].....30
Tabelle 6: Validatoren [McKay].....31

1 Einführung

Dieser Abschnitt soll einen kurzen Überblick über den Inhalt und das Ziel der Studienarbeit geben.

1.1 Gegenstand der Studienarbeit

Die Studienarbeit beschäftigt sich mit dem Internetportal der European Social Simulation Association (ESSA). Dieses Portal stellt eine Plattform für den Informationsaustausch der Mitglieder von ESSA dar und soll gleichzeitig Informationen über die Entwicklung, Forschung, Lehre und Anwendung von sozialwissenschaftlichen Simulation bereitstellen.

1.2 Zielsetzung

Ziel der Studienarbeit ist es, das bestehende Portal auf ein neues, zukunftssicheres Content-Management-System (CMS) zu migrieren und die vorhandenen Inhalte in das neue System zu integrieren. Im Rahmen der Migration des CMS der Universität Koblenz-Landau von Zope zu Plone (ein auf Zope aufsetzendes CMS) ist die zukünftige Systemumgebung vorgegeben.

Darüber hinaus soll das Portal um einige Punkte erweitert werden:

1. Integration eines Wikis, in dem Fachbegriffe der sozialwissenschaftlichen Simulation erläutert werden.
2. Integration einer Bibliographie, in der fachspezifische Literatur aufgelistet wird.
3. Einbindung eines Kurskataloges (SocSimNet), in dem fachspezifische Veranstaltungen, Dozenten und Veranstaltungsorte verwaltet werden.

Da die Umstellung des CMS an der Universität Koblenz-Landau noch nicht begonnen wurde, dient die Studienarbeit gleichzeitig als Indikator für eventuell auftretende Entwicklungsprobleme und Fallstricke beim Einsatz von Plone. Aus diesem Grund wird der Implementierung des Kurskataloges ein eigener Abschnitt gewidmet, der auf die Programmierung einer Zusatzapplikation (Produkt) in Plone im Detail eingeht und auch die Probleme bei deren Entwicklung aufzeigt.

Auf die komplette grundlegende Arbeitsweise mit Plone kann in dieser Arbeit nur in Grundzügen eingegangen werden, für Details sein dazu auf das Plone-Buch [McKay] verwiesen. Eine gute „Grobübersicht“ bietet auch [7].

2 Begriffe

In diesem Kapitel werden die wichtigsten Begriffe im Zusammenhang mit dieser Arbeit erläutert.

2.1 Portal

Der Ausdruck Portal (lat. *porta*, „Pforte) bezeichnet in der Informatik einen zentralen Zugang, über den man auf individuell zugeschnittene, unternehmensinterne und externe Informationen und Dienste zugreifen kann. Ein Webportal ist ein vereinfachter Spezialfall davon, der eine spezielle Form einer Internetseite darstellt, die in der Regel als Startseite zu einem bestimmten Thema gestaltet ist. [vgl. 8]

Die Internetseite der ESSA kann man als personalisiertes, vertikales Webportal bezeichnen. Es zielt auf eine relativ kleine Zielgruppe, welche gleichzeitig den Inhalt der Seite beeinflussen kann. Im Gegensatz dazu stehen horizontale Portale, die eine breite Zielgruppe abdecken sollen und nicht-personalisierte, deren Inhalt nicht vom Benutzer beeinflusst werden kann. Beispiele für ein horizontale, nicht-personalisierte Portale sind Suchmaschienen (z.B. Google) oder Online-Telefonbücher.

2.2 Wiki

Auch WikiWiki (hawaiianisch für „schnell“) oder WikiWeb genannt, stellen Wikis im Internet verfügbare Sammlungen von Seiten dar, die von Benutzern nicht nur gelesen, sondern auch geändert werden können. Die Seiten eines Wiki sind durch Querverweise miteinander verbunden. Das wahrscheinlich bekannteste Wiki ist Wikipedia (www.wikipedia.org), eine Online-Enzykopädie.

Entstanden sind Wikis als Wissensmanagement-Tools nach einer Idee aus dem Jahr 1995. Das erste WikiWikiWeb wurde in diesem Jahr von Ward Cunningham entwickelt. Wikis verwirklichen eine der Visionen von Tim Berners-Lee, dem Erfinder des World Wide Web, in der Informationen online verfügbar und auch direkt bearbeitbar sein sollen. [vgl. 4]

Wikis benutzen in der Regel eine einfache Syntax zum Formatieren der eingestellten Informationen. Diese Syntax ist jedoch wesentlich einfacher aufgebaut als zum Beispiel HTML, um auch Laien einen einfachen Zugang zu den Systemen zu ermöglichen.

Neuere Entwicklungen versuchen Wikis untereinander mit so genannten InterWiki-Links zu verbinden. Diese Techniken sind zum aktuellen Zeitpunkt aber als proprietär einzustufen, da diverse Probleme, zum Beispiel die Verwaltung einer weltweiten Liste aller Wikis durch eine zentrale Registrierungsstelle, noch nicht gelöst sind.

2.3 Zope

Zope ist ein Webanwendungsserver (Application Server). Zope steht für Z Object Publishing Environment. Er ist Open Source und (bis auf eine performance-kritische Teile) in Python programmiert, und damit vollständig Objektorientiert.

Er kann durch so genannte Produkte erweitert werden, die einfach durch kopieren in ein Unterverzeichnis installiert werden können. Diese Produkte können z.B. Newsletter-Systeme, Gästebücher oder auch komplette Content-Management-Systeme (siehe Plone) sein.

Zope besitzt eine eingene, proprietäre Datenbank, die ZODB (Z Object Data Base), in der alle Inhalte abgelegt werden. Zusätzlich zu dieser Datenbank können auch andere relationale Datenbanken verwendet werden, um die Objekte zu speichern.

Die Objektorientierung ist zentraler Bestandteil von Zope. Sämtliche Inhalte werden in Objekten abgelegt. Die Objekte können Eigenschaften und Methoden erben, was die Entwicklung von neuen Inhaltstypen vereinfacht. Objekte können jedoch nicht nur Inhalte sein, sondern auch Python-Skripte, welche in der ZODB gespeichert werden und in einer Art „Sandbox“ mit beschränkter Rechten ausgeführt werden können.

Es verfügt über drei Entwicklungssprachen, in der Inhalte bzw. Layouts definiert werden können. Dies sind DTML (Document Template Markup Language), Python und ZPT (Zope Page Templates). Die Verwendung von Python in Kombination mit ZPT ermöglicht eine Trennung der Programmlogik vom Inhalt und ist deshalb der Entwicklung mit DTML vorzuziehen.

Zope baut auf einem rollenbasierten Sicherheitssystem auf, welches mit Benutzern und Gruppen arbeitet und sich sehr feingranular einstellen lässt.

Mit Zserver bringt Zope einen eigenen Webserver mit, welcher eine erweiterte Version des in Python programmierten Medusa ist. Oft wird dieser hinter einem

als Proxy konfigurierten Apache-Webserver verwendet; dies ist auch sehr sinnvoll, da der mitgelieferte Webserver keinerlei Sicherheits- oder Anfrageüberprüfungsfunktionen besitzt.

2.4 Plone

Plone ist ein Content-Management-System, welches in Python programmiert ist und auf Zope (zusammen mit dessen Content Management Framework) aufsetzt.

Es bringt von sich aus eine mehrsprachige Oberfläche, ein Standardtemplate und einen WYSIWYG-Editor mit. Standardmäßig kennt Plone folgende Inhaltstypen:

- Dokument
  Ein Dokument stellt irgendwelche Informationen dar. Dabei kann ein Dokument ein bestimmter Typ sein (Nachricht, Bild, ...) und auch wieder andere Dokumente enthalten.
- Nachricht
  Eine Nachricht ist ein Dokument, welches in automatisch unter dem Nachrichten-Reiter von Plone erscheint.
- Link
  Beschreibt einen Verweis auf ein anderes Element. Dieses Element kann ein anderes Dokument oder eine externe Internetseite sein.
- Bild
  Bilder aller Art, z.B. JPG- oder GIF-Dateien
- Termin
  Termine werden automatisch im standardmäßig in Plone vorhandenen Kalender angezeigt.
- Ordner
  Ein Ordner entspricht einen Verzeichnis auf der Festplatte und dient der Strukturierung der Inhalte. In ihm können andere Dokumente abgelegt werden.
- Thema
  Mit einem Thema (auch intelligenter Ordner genannt) werden Inhalte gruppiert. Es stellt im Grunde genommen ein gespeichertes Suchkriterium dar, welches später wiederverwendet werden kann.

- Datei

  Eine Datei ist irgend ein beliebiger Inhalt, der auf die Seite hochgeladen werden kann.

Dokumente in Plone können über einen Workflow freigegeben werden, so dass Ihnen je nach aktuellem Zustand unterschiedliche Zugriffsberechtigungen und Sichtbarkeiten zugeordnet werden können.

Für Plone gibt es eine große Anzahl an Produkten, durch die es erweitert werden kann. Darunter fallen Fotoalben, Gästebücher oder die Unterstützung mehrsprachiger Dokumente durch PloneLingua.

Plone verwendet ein Framework (Archetypes), welches die Erstellung neuer Inhaltstypen und Produkten vereinfachen soll. Dafür stehen auch weitere Hilfsmittel zur Verfügung, wie z.B. ArchGenXML, mit dem sich Produkte aus UML-Diagrammen erstellen lassen. Genaueres hierzu stehen im Kapitel 5, da die Kursverwaltung der ESSA-Seite auf diese Art erstellt werden soll.

2.5 Content-Management-System (CMS)

Ein Content-Management-System dient zur (gemeinschaftlichen) Erstellung und Bearbeitung von Dokumenteninhalten (dem Content) durch ein Anwendungsprogramm. Der Content kann dabei sowohl Text als auch multimedial (Bilder, Videos, ...) sein. Dabei wird versucht, den reinen Inhalt von dessen Darstellung zu entkoppeln.

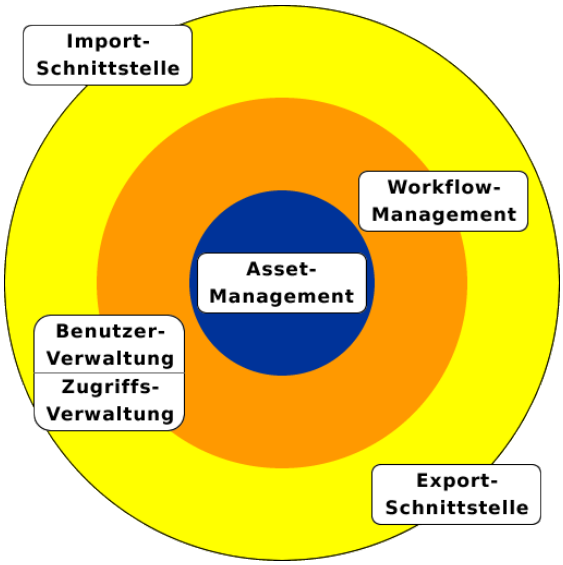

Abbildung 1: Aufbau eines CMS [6]

Die Entkopplung von Inhalt und Darstellung hat den Vorteil, dass Inhalte auf verschiedenen Ausgabemedien (z.B. als HTML- oder PDF-Datei) ausgegeben werden können, indem sie mit Vorlagen (Templates) gekoppelt werden.

Der Grundlegende Aufbau eines CMS geht aus Abbildung 1 hervor. In der Regel besitzen sie ein Asset-Management zum Verwalten der Inhalte, ein Workflow-Management, um den Dokumentenfluss zu steuern, Benutzer- und Zugriffsverwaltung, um den Zugriff auf einzelne Inhalte einzuschränken und diverser Import- und Exportschnittstellen für Inhalte.

Speziell auf Internetseiten benutzt man Content-Management-Systeme zur Verwaltung der Zugriffsrechte einzelner Benutzer zum Erstellen, Bearbeiten und Lesen von Dokumenten und zum Life-Cycle-Management, der zeitgesteuerten Verfügbarkeit einzelner Dokumente. Sehr oft kommen hier noch Versions-Management-Funktionen hinzu, um Veränderungen einzelner Inhalte zu Dokumentieren.

3 Anforderungen

Das bestehende Portal basiert auf PHPNuke, einem in PHP programmierten, sehr einfachem Content-Management-System, welches auf dem Webserver der Universität Koblenz-Landau gehostet wird und unter der Internetadresse http://www.essa.eu.org/ erreichbar ist. Dieses Portal wurde 2003 im Rahmen einer Studienarbeit von Bogdan Werth und Thomas Krämer unter Betreuung durch Prof. Dr. Klaus G. Troitzsch entwickelt.

Abbildung 2: Layout der bestehenden ESSA-Seite

Es besitzt eine Startseite, auf der die wichtigsten und aktuellsten Themen zusammengefasst sind, einen Kalender, einen Newsletter, eine Benutzerverwaltung und eine Linksammlung. Neue Benutzer können sich über ein Anmeldeformular auf der Seite registrieren.

Dieser Abschnitt beschreibt die Anforderungen an das neue Portal und vergleicht diese mit den Anforderungen des Ursprünglichen aus der Arbeit von Bogdan Werth und Thomas Krämer. Anforderungen, die als Kann-Anforderungen gekennzeichnet sind, wären für das ursprüngliche System wünschenswert gewesen, sind aber nicht umgesetzt worden. Dazu werden zuerst die Original-Anforderungen wiedergegeben und dann jeweils kurz auf die Besonderheiten der Umsetzung in Plone eingegangen.

3.1 Gleichgebliebene Anforderungen

Die hier aufgeführten Anforderungen sind seit Implementierung des ursprünglichen Systems gleich geblieben.

3.1.1 Benutzerfreundlichkeit

„Alle vom Portal angebotenen Dienste sollten möglichst intuitiv und ohne Studium der Anleitung zu bedienen sein. Insbesondere sollte es möglich sein, Artikel und sonstige Texte auch ohne Kenntnisse in HTML zu verfassen. Es sollte ebenfalls eine möglichst einheitliche Oberfläche und Bedienung der Funktionen gewährleistet sein. Kann dies nicht überall erreicht werden, so sollte an den jeweiligen Stellen eine Hilfefunktion angeboten werden.“ [WerthKrämer]

Diese Anforderung erfüllt Plone in mehrfacher Hinsicht von sich aus, da es für Benutzer sehr einfach zu verwenden ist, viele Menupunkte in verschiedenen Sprachen zur Verfügung stehen, ein einfach zu bedienender WYSIWYG-Editor bereits mitgeliefert wird und an vielen Stellen eine mehrsprachige Hilfe direkt bei der Eingabe als Hinweistext zur Verfügung steht.

3.1.2 Möglichkeit zur Veröffentlichung von eigenen Artikeln

„Alle berechtigten Benutzer sollen in der Lage sein, eingene Artikel in das Portal einzustellen. Alle Artikel sollen unter einer auszuwählenden Kategorie zusammengefasst und unter dieser abgerufen werden können. Die Artikel sollen allen Besuchern der Seite zur Verfügung stehen.“ [WerthKrämer]

Diese Anforderung erfüllt sich auf triviale Weise, da Plone als Content-Management-System genau für diesen Zweck entworfen wurde. Jeder Benutzer hat ein eigenes Heimatverzeichnis, in dem er Inhalte erstellen kann, die dann über den Workflow von einem berechtigten Benutzer für alle Besucher veröffentlicht werden können.

3.1.3 Möglichkeit zur Führung eines Veranstaltungskalenders

„Bedingt durch die Zielsetzung der ESSA, Termine der verschiedenen Kongresse und anderer Veranstaltungen rund um das Thema sozialwissenschaftliche Simulation zu koordinieren, soll das Portal die Führung eines Veranstaltungskalenders ermöglichen. Jeder Besucher des Portals soll in der Lage sein, bestehende Einträge anzusehen und neue zu melden. Ein gemeldeter Termin soll zunächst durch einen dafür berechtigten Benutzer geprüft und dann freigeschaltet werden können.“ [WerthKrämer]

Plone bringt von sich aus einen Kalender mit, der standardmäßig schon aktiviert ist. Benutzer können zum Eintragen diese in ihrem Heimatverzeichnis erstellen und zur Veröffentlichung vorschlagen.

3.1.4 Verzeichnis aller Mitglieder

„Alle Nutzer, sowohl Besucher als auch registrierte Mitglieder, sollen die Möglichkeit haben, eine Übersicht der bereits registrierten Mitglieder zu erhalten. Dabei soll eine einfache Möglichkeit zur Kontaktaufnahme via E-Mail angeboten werden.“ [WerthKrämer]

Alle Benutzer können über eine in Plone integrierte Mitgliedersuche gefunden werden. Diese ist auch in der Lage, sämtliche Benutzer aufzulisten. Darüber besitzt jedes Mitglied eine eigene Infoseite, auf der die wichtigsten Informationen incl. E-Mail-Adresse aufgeführt sind.

3.1.5 Möglichkeit zur Organisation von Links

„Um auf interessante Inhalte und deren Angebote im Internet, die das Thema sozialwissenschaftliche Simulation betreffen, aufmerksam machen zu können, soll das Portal eine Linksammlung anbieten. Die Links sollen nach Kategorien geordnet werden können. Jeder Besucher des Portals soll einen neuen Link melden können, der dann zunächst durch einen dafür berechtigten Benutzer geprüft und dann freigeschaltet werden kann.“ [WerthKrämer]

In Plone kann jedes Mitglied Links in seinem Heimatverzeichnis erstellen und zur Veröffentlichung vorschlagen. Diese Links können mit Stichwörtern versehen werden, so dass sie über Themen/Intelligente Ordner auf der Seite angezeigt werden können.

3.1.6 Möglichkeit zum Anbieten von Downloads

„Zur Bereitstellung von Dateien, die heruntergeladen werden können, soll das Portal die Möglichkeit bieten, eine Downloadsammlung zu organisieren. Die einzelnen Downloads sollen wiederum nach Kategorien geordnet werden können. Neue Downloads, die jeder Besucher des Portals melden kann, sollen zunächst durch einen dafür berechtigten Benutzer geprüft und dann freigeschaltet werden können.“ [WerthKrämer]

Jeders Mitglied kann Dateien in sein Heimatverzeichnis hochladen und diese zur Veröffentlichung vorschlagen. Diese Dateien können auch beliebig in andere Dokumente integriert werden.

3.2 Entfernte Anforderungen

Die in diesem Abschnitt beschriebenen Anforderungen sollen aus dem System entfernt werden.

3.2.1 Kommunikationsmöglichkeit

„Das Portal sollte den Mitgliedern der ESSA eine einfache Möglichkeit zur Kommunikation bieten. Diese sollte sich möglichst gut in die übrigen Dienste des Portals integrieren. Sie ist lediglich als internes Nachrichtensystem zu konzipieren. Die externe Kommunikation erfolgt weiterhin über die bereits etablierten E-Mail-Systeme der einzelnen Mitglieder. Des weiteren sollen sowohl private als auch öffentliche Mailinglisten zur Verfügung stehen.“ [WerthKrämer]

Diese Anforderung wurde entfernt, da das in PHPNuke vorhandene Kommunikationssystem nicht verwendet wurde. Die Kommunikation der Nutzer untereinander soll mittels normaler E-Mail stattfinden.

3.2.2 Möglichkeit zur Führung eines Journals

„Zur Dokumentation der eigenen Tätigkeiten und zur Bereitstellung und Verbreitung von Inhalten innerhalb der Organisation sollte das Portal die Möglichkeit bieten, ein Journal zu führen. Alle Einträge dieses Journals sollten öffentlich und somit allen Mitgliedern der ESSA zugänglich sein.“ [WerthKrämer]

Die Notwendigkeit eines gesonderten Journals entfällt, da in Plone jedes Dokument eine Historie besitzt und zusätzlich eine globale Liste aller zuletzt veränderten Dokumente geführt wird, die für alle Mitglieder sichtbar ist.

3.2.3 Hilfe-System (Kann-Anforderung)

„Es wird gewünscht, ein vollständiges Hilfe-System anzubieten, das alle möglichen Funktionen beschreibt und überall angebunden werden kann. Es soll entweder in Form eines neuen Fensters oder als Popup realisiert werden.“ [WerthKrämer]

Erste Versuche mit Plone haben gezeigt dass ein damit erstelltes Portal sich so intuitiv verwenden lässt, dass eine zusätzliche Online-Hilfe nicht erforderlich ist.

Das Hilfe-System ist auch im ursprünglichen System nicht umgesetzt worden. Für einige komplexere Vorgänge im administrativen Bereich wird später noch ein Handbuch geschrieben, welches z.B. die Erstellung eines Newsletters oder einer Umfrage im Detail erläutert.

3.2.4 Favoriten für das Journal (Kann-Anforderung)

„Im Zuge der Entwicklung ergab sich der Wunsch, im Journal-System die Möglichkeit zu implementieren, das Journal bestimmter Mitglieder zu persönlichen Favoriten zu nehmen, um den Zugriff darauf abzukürzen.“ [WerthKrämer]

Die Anforderung ist auch im Original-System nicht umgesetzt worden. Plone kann dies jedoch von sich aus bewerkstelligen, da es eine persönliche Favoritenliste für jede angemeldete Person mitbringt, die standardmäßig deaktiviert ist, sich aber durch geringen Aufwand des Administrators aktivieren lässt. Es ist geplant, diese Funktionalität später bei Interesse der Benutzer zu aktivieren. Eine Beschreibung der Favoriten incl. Bildern ist in [7] zu finden.

3.3 Neue Anforderungen

Alle hier aufgeführten Anforderungen sollen im neuen System umgesetzt werden.

3.3.1 Einbindung eines Wiki zur Verwaltung von Begriffen aus dem Bereich sozialwissenschaftliche Simulation

Es soll ein Wiki eingebunden werden, in dem Erklärungen zu Begriffen aus dem Bereich der sozialwissenschaftlichen Simulation erläutert werden. Angemeldete Benutzer sollen Einträge in diesem Wiki vornehmen können. Da Wikis von sich aus eine Historie der jeweiligen Einträge speichern, ist eine Freischaltung durch einen berechtigten Benutzer hier nicht notwendig.

3.3.2 Integration einer Bibliographie zur Verwaltung von Literatur, die sich mit dem Thema sozialwissenschaftliche Simulation beschäftigt

Es soll die Möglichkeit geschaffen werden, Literatur zu erfassen, die sich mit dem Thema sozialwissenschaftliche Simulation beschäftigt. Jeder Benutzer soll die Möglichkeit haben, neue Literatur zu melden, die nach Prüfung durch einen berechtigten Benutzer für alle sichtbar geschaltet wird.

3.3.3 Einbindung eines Kurskataloges (SocSimNet) zur Verwaltung von Kursangeboten, Dozenten und Veranstaltungsorten

Der bisher als eigenständige Internetseite verfügbare Kurskatalog soll in die Seite integriert werden. Zu diesem Zweck soll ein Plone-Produkt entworfen werden, welches nach Möglichkeit die bestehende Datenbank einbinden kann. Genauere Anforderungen an dieses Produkt und Informationen folgen in Kapitel xxx.

3.3.4 Bindung an das Zielsystem Plone

Das Zielsystem, auf dem das Portal neu implementiert werden soll ist von Beginn an Vorgegeben. Die Verwendung von Plone wurde festgelegt, da die Universität Koblenz-Landau in absehbarer Zeit auf dieses System umgestellt werden soll und die ESSA-Seiten dort gehostet werden. Dies führt dazu, dass Systemupdates und Backup automatisiert über das Rechenzentrum erfolgen können. Eine genaueer Beschreibung der Zielumgebung ist in Kapitel 4 zu finden.

3.4 Zusammenfassung

Die grundlegenden Anforderungen an das Portal haben sich nicht sehr stark geändert. Es handelt sich hauptsächlich um Erweiterungen der Funktionalität, um bisher nicht integrierbare Informationen in strukturierter Weise in die Seite mit einbinden zu können. Des weiteren Fallen auch einige undokumentierte Anforderungen weg, wie zum Beispiel die Verwaltung der Mitgliederbeiträge durch den Schatzmeister Prof. Dr. Klaus G. Troitzsch, die einmal über die Mitgliederverwaltung in PHPNuke und ein zweites mal über externe Tools gehandhabt wurde.

4 Implementierung des neuen Systems

In diesem Abschnitt wird die Installation von Zope und Plone kurz erläutert, die Anpassungen der Templates für das Erscheinungsbild der Seite und die Installation des Wiki-, Bibliographie- und Newsletter-Produkts beschrieben.

Die genaue Zielumgebung bis zum Abschluss der Studienarbeit noch nicht feststand, wurde das System zuerst in einer Produktionsumgebung implementiert, deren Inhalt später in das endgültige System migriert wird. Da sowohl Zope als auch Plone Betriebssystemunabhängig arbeiten, wurde für die Produktionsumgebung ein Debian 3.1 Sarge System gewählt, welches Python in der Version 2.4.1 und C in der Version 3.3.5 einsetzt. Zope wurde in der Version 2.9.2 installiert, Plone in der Version 2.1.2.

4.1 Installation von Zope und Plone

Zuerst wird Zope installiert und gestartet. Darauf soll hier nicht eingegangen werden, es sei statt dessen auf [McKay] verwiesen.

Im Anschluss daran wird Plone installiert. Dazu wird die die Quellcode-Datei von http://www.plone.org/download heruntergeladen und in das Products-Verzeichnis der Zope-Instanz entpackt. Nach einem Neustart von Zope steht im ZMI (Zope Management Interface) im Root-Verzeichnis der Eintrag „Plone Site“ zur Verfügung, mit dessen Hilfe eine neue Plone-Instanz angelegt wird. Diese wird ESSA genannt. Danach ist die Instanz einsatzbereit.

Als erstes muss man sich an der Plone-Instanz als Benutzer registrieren. Nach erfolgter Registrierung ist man ein normaler Benutzer, der das Recht hat, Inhalte hinzuzufügen (zuerst aber nur in seinem Heimatverzeichnis).

Alle weiteren Produkte, die benötigt werden, werden in das Products-Verzeichnis der Zope-Instanz kopiert und können danach aus dem Plone-Control-Panel selber von einem Benutzer mit Manager-Rechten aktiviert werden. Dazu bietet die Plone-Konfigurationsseite eine Liste von installierten und noch installierbaren Produkten an.

Die für die Implementierung notwendigen Produkte Archetypes 1.3.7-final, ATContentTypes 1.0.3-final, ATReferenceBrowserWidgets 1.1, ResourceRegistries 1.1, MimeTypesRegistry 1.3.8-final2 und PortalTransforms 1.3.9-final2 sind in Plone mitgeliefert und schon aktiv, die ATExtensions 0.7.1

müssen noch über die Konfigurationsseite aktiviert werden. Zusätzlich musste in der Produktionsumgebung noch ArchGenXML 1.4.0-beta2 installiert werden, um das SocSimNet-Produkt zu generieren.

4.2 Grundeinstellungen vornehmen

Über das Plone-Control-Center werden zuerst einige Grundeinstellungen vorgenommen. Unter „E-Mail-Einstellungen“ wird der Mailserver konfiguriert, über den Mails verschickt werden sollen. Bei „Website Allgemein“ werden der Name der Seite, eine kurze Beschreibung, ein Absender für automatisch verschickte E-Mails, Standardsprache, Vergabe von Passwörtern bei Registrierung, Konfiguration externer Editoren und kurzer Artikelnamen eingestellt.

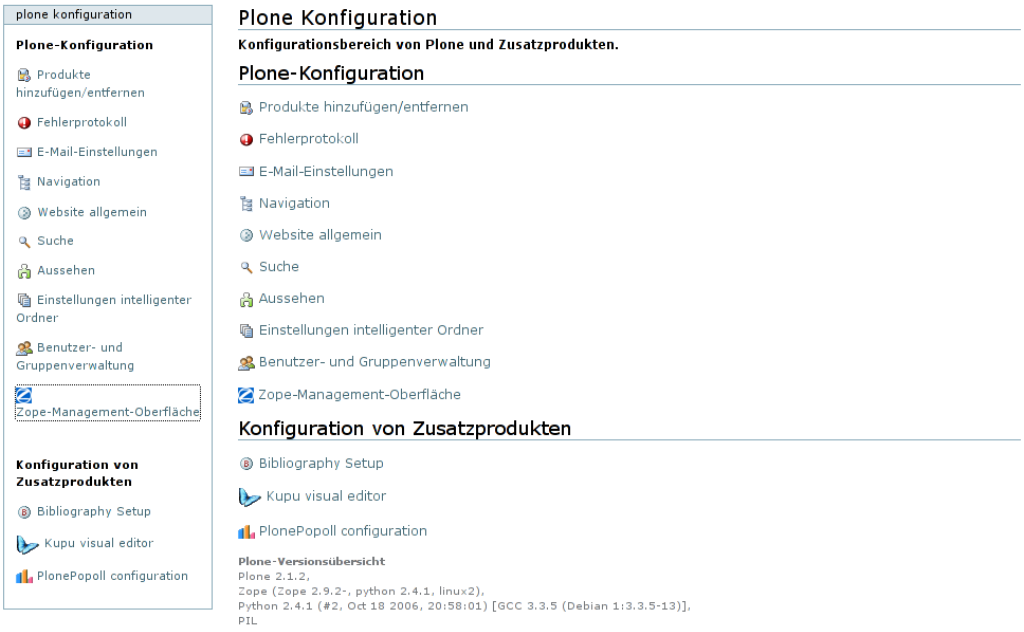

Abbildung 3: Plone Control Center

Die Einstellungen zu Fehlermeldungen werden unter gleichnamigem Menupunkt vorgenommen, wobei das nachträglich installierbare PloneErrorReporting-Produkt hilft, diese in verständlicher Form anzuzeigen.

4.3 Anpassung der Templates

Zum Anpassen des Erscheinungsbildes gibt es zwei Möglichkeiten. Zum einen kann man ein komplettes Template über das Plone-Control-Panel unter „Aussehen“ aktivieren, zum anderen kann man das vorhandene Standard-Template über das ZMI anpassen.

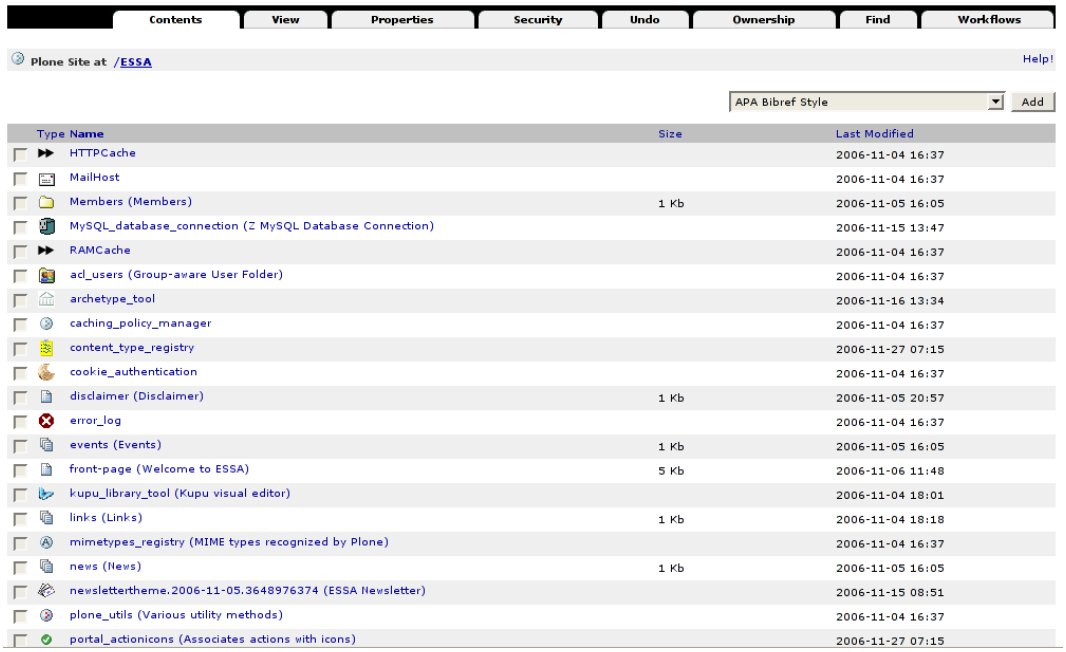

Abbildung 4: ZMI (Zope Management Interface)

Da das optische Erscheinungsbild des Standard-Templates sehr nah an das gewünschte Endergebnis herankommt, soll dieses angepasst werden. Eines der größten Vorteile des Standard-Templates ist außerdem die Standardkonformität wie auch die Barrierefreiheit.

Zum Verständnis der Anpassungen müssen erst einige grundlegende Eigenschaften des Standard-Templates erläutert werden. Eine Plone-Seite besteht aus drei Spalten, einer linken und rechten zur Darstellung so genannter Portlets, kleiner Kästen zur Anzeige von Informationen (Navigation, Kalender, ...), und der mittleren, die das aktuelle Objekt anzeigt. Die Portlets haben in der Regel eine eigene Logik, die bestimmt wann und wie sie angezeigt werden. Die Anzeigereihenfolge der Portlets wird über Eigenschaften left_slots und right_slots im ZMI unter dem Properties-Reiter des Plone Wurzelverzeichnisses eingestellt.

Die Templates selber werden im ZMI unter portal_skins geändert, diese sind jedoch geschützt, es wird aber automatisch eine Kopie von ihnen angelegt, die dann verwendet wird.

Ein Template ist eine XHTML-Datei, die aus HTML-Bereichen besteht und mit TALES (Template Attribute Language Extression Syntax) und externen CSS (Cascading Style Sheets) angereichert wird.

Zum Ändern des Logos wird im ZMI portal_skins, dann plone_images und schließlich logo.jpg aufgerufen. Nach Anklicken von Customize wird das Objekt in den custom-Ordner kopiert, da der Original-Skin geschützt ist. Das gleiche passiert mit dem Footer, welcher im Ordner plone_templates zu finden ist. In diesem wird das Attribut tal:condition="nothing" in den umgebenden div hinzugefügt, damit wird er ausgeblendet.

Abbildung 5: Neues Design der ESSA-Seite

4.4 Wiki

Als Wiki-Software wurde ZWiki in der Version 0.55.0 gewählt, dieses ist ursprünglich für Zope entwickelt worden und bringt zugleich eine passende Oberfläche für Plone mit. Die Installation verläuft hier nicht ganz so trivial wie bei einem normalen Plone-Produkt, da ZWiki keine „automatische" Installation als komplettes Wiki in Plone anbietet.

Zuerst wird ZWiki in das Products-Verzeichnis der Plone-Instanz kopiert und Zope neu gestartet. Danach steht es in Plone-Konfigurationsmenu unter Produkte zur installation. Nachdem es dort installiert wurde, kann man leider nicht direkt ein Wiki anlegen, sondern muss einen Umweg über das ZMI gehen. Zuerst legt man in Plone einen einfachen Folder an (hier Wiki), in dem das Wiki später liegen soll. Danach legt man im ZMI ein neues ZWiki im Root-Verzeichnis an, gibt ihm aber noch einen anderen Namen (hier testwiki). In diesem Verzeichnis wird nun

ein komplettes Wiki angelegt. Daraufhin kopiert man alle Inhalte des testwiki-Ordners in den Wiki-Ordner und der testwiki-Ordner kann wieder entfernt werden.

Damit beim Aufruf einer Wiki-Seite nicht alle Einträge im Menu aufgeführt werden, wird in der Plone-Konfiguration unter Navigation der Eintrag Wiki-Page entfernt.

Abbildung 6: Startseite des Wikis

Die Erstellung neuer Seiten erfolgt dabei mit Hilfe von Structured Text, welches einfach eine Formatierung ohne Programmierkenntnisse zulässt. Dabei werden sehr einfache Steuerzeichen verwendet. Ein einzeiligere Paragraph, dem ein längerer mit einer Einrückung der ersten Zeile folgt, wird automatisch zu einer Überschrift. Zeilen, die mit Gedankenstrich oder einer Zahl gefolgt von einem Leerzeichen beginnen, werden zu Aufzählungen. Fettschrift wird durch Umrahmen mit zwei Sternen (**...**), Kursivschrift durch Umrahmen mit einem Stern (*...*), Unterstreichungen durch Umrahmen mit Unterstrichen (_..._) und Text mit konstanter Zeichenbreite durch Umrahmen mit einfachen Hochkommata ('...').

Bekannte Wörter werden automatisch durch Links innerhalb des Wikis ersetzt, möchte man automatisch bei unbekannten Wörtern den Erstellungs-Link angezeigt bekommen, muss man diese mit eckigen Klammern umrahmen. Interessant ist, dass ZWiki über InterWikiLinks verfügt, diese werden durch Name_des_Wikis:Term dargestellt. Die Liste der unterstützten Wikis ist unter http://zwiki.org zu finden und wird ständig erweitert.

Zusätzlich unterstützt es normale HTML-Tags auf den Seiten, so dass erfahrenere Anwender zusätzliches Layout in Ihre Artikel einbringen können.

4.5 Bibliographie

Zur Verwaltung Bibliographischer Einträge wurde CMFBibliographyAT installiert. Dieses Produkt hat bringt von sich aus Vorlagen für fast alle möglichen Publikationen wie z.B. Zeitungsartikel, Buch, usw. mit. Literatur kann von jedem Benutzer des Systems in seinem Heimatverzeichnis erstellt werden und zur Veröffentlichung eingereicht werden. Nach Freigabe durch einen Manager ist der Literatureintrag über eine zentrale Seite (Bibliographie) sichtbar, die automatisch in der oberen Navigationsleiste der Seite installiert wird.

CMFBibliographyAT lehnt sich von seinem Schema sehr an BibTeX an, so dass hierfür Import- und Exportfunktionen zur Verfügung stehen (wobei es noch bekannte Fehler gibt, wenn diese von EndNote generiert wurden). Viel mehr ist zu diesem Produkt auch nicht zu sagen, da sowohl die Inhaltstypen als auch die Funktionsweise selbsterklärend sind, da sie wie ganz normale Plone-Inhalt angelegt und verwaltet werden.

4.6 Newsletter

Als Newsletter-Produkt wurde PloneGazette in der Version 1.1 gewählt. Dieses machte bei den Tests den stabilsten Eindruck und kann auch sehr stark skalieren, wenn das MaildropHost-Produkt installiert ist. Nachteil des Produktes ist allerdings, dass Lynx, welches für alle Linux-Distributionen erhältlich ist, aber den Einsatz auf einem Windows-System erschweren würde, auf dem System installiert ist.

Nach der gewohnten Installation mittels Plone-Konfiguration stehen mehrere neue Inhaltstypen zur Verfügung, die „Newsletter Theme“, der „Newsletter Large Folder“ und ein Abonnenten. Zuerst wird im Root-Verzeichnis eine neue Newsletter Theme angelegt, es wird sofort nach diversen Einstellungen gefragt. In

dieser Newsletter Theme legt man einen „Newsletter Large Folder“ an, in dem später die Abonnenten gespeichert werden. Dieser wird in den Eigenschaften der Newsletter Theme eingetragen; wichtig hierbei ist, dass die ID des Folders und nicht dessen Name erwartet wird. Wichtig ist auch, in der Plone-Konfiguration die Anzeige der Newsletter Large Folder in den Navigationseinstellungen auszuschalten, da ansonsten jeder Benutzer diesen zwar in der Navigation sieht, aber nur eine Fehlermeldung beim Aufruf sieht.

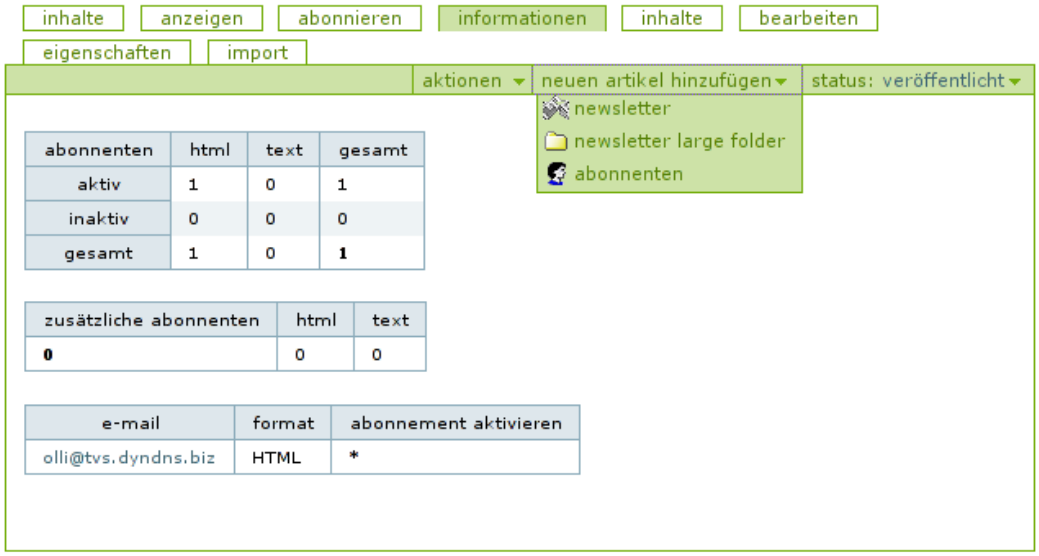

Abbildung 7: Newsletter - Abonnenten Verwalten

Alle Benutzer der Seite können sich registrieren, um den Newsletter zu erhalten. Dabei können Sie festlegen, ob sie den Newsletter in HTML oder in PlainText bekommen möchten.

Zum Anlegen eines Newsletters wird über „neuen artikel hinzufügen“ ein neuer Newsletter angelegt, danach öffnet sich die Seite zum Editieren. Hier können die Haupttext über den WYSIWYG-Editor eingegeben werden und ein Absendezeitpunkt eingetragen werden (ein zeitgesteuertes automatisches Versenden hat aber zumindest auf dem Testsystem nicht funktioniert). Interessant wird es aber eigentlich erst, wenn der Haupttext des Newsletters gespeichert ist, danach bietet sich nämlich die Möglichkeit, so genannte Newsletter Topics und Newsletter References hinzuzufügen. Topics sind hier wie in Plone auch intelligente Suchanfragen, mit denen vorhandene Plone-Objekte anhand ihren Metatyps und anderen Suchkriterien automatisch an den Newsletter angehängt werden können. Mit Newsletter References können einzelne Plone-Objekte zum

Newsletter hinzugefügt werden. Zusätzlich lassen sich die Topics und Referenzen noch in Ordnern strukturieren.

Des weiteren bietet das Produkt eine Vorschau, in der der Newsletter sowohl in HTML- als auch im Textformat vor dem Versenden betrachtet werden kann und an den Newsletter-Administrator (leider läßt sich nur einer eintragen) versendet werden kann.

4.7 Umfrage-Tool

Als Umfrage-Tool wurde PoPoll in der Version 2.5.1 gewählt. Bei diesem mussten jedoch die Templates angepasst werden, da sonst auch anonyme Besucher die Abstimmungen zu sehen bekommen hätten.

Eine Umfrage kann von jedem Benutzer in seinem Heimatverzeichnis angelegt werden und zur Veröffentlichung eingereicht werden. Danach sind die Umfragen über einen Link erreichbar bzw. in einem Portlet sichtbar. Welche Umfrage im Portlet sichtbar sein soll, kann von einem Manager in der Plone-Konfiguration eingestellt werden.

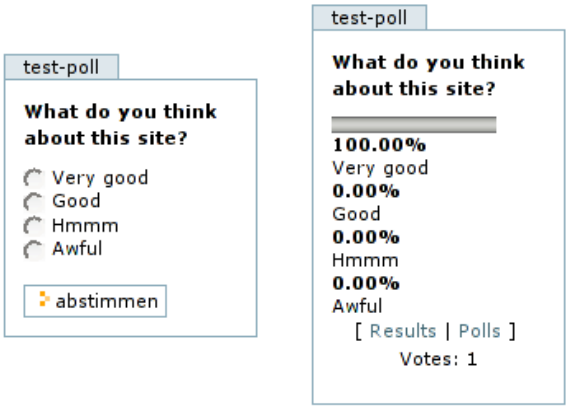

Abbildung 8: Poll - Abstimmung und Ergebnis im Portlet

Da im Auslieferungszustand das Umfrage-Portlet für alle Benutzer sichtbar ist, musste dieses angepasst werden, damit anonyme Benutzer nicht an der Umfrage teilnehmen können. Dazu wurde im ZMI eine Kopie der Datei portlet_popoll im portal_skins-Ordner unter PlonePopoll angelegt (diese wird wieder automatisch in den custom-Ordner verschoben), in der folgende Änderungen gemacht wurden:

Der Bereich

```
<div metal:define-macro="portlet"
     tal:define="polls python:here.PlonePopoll_getPortletPoll();"
     tal:condition="polls">
```

am Anfang der Datei wird dazu geändert in:

```
<div metal:define-macro="portlet"
     tal:define="polls python:here.PlonePopoll_getPortletPoll();"
     tal:condition="python:polls and not
     here.portal_membership.isAnonymousUser()">
```

Eine Umfrage besteht dabei immer aus genau einer Frage und mehreren möglichen Antworten. Es kann ausgewählt werden, wie viele Antworten man wählen darf. So sind Umfragen mit einer einzigen Wahlmöglichkeit möglich (z.B. Wahlen einer Person aus mehreren oder ja/nein-Umfragen), als auch Umfragen mit mehreren möglichen Antworten (z.B. „Wähle zwei mögliche Termine für ein Treffen). Zusätzlich läßt sich einstellen, ob die Benutzer sofort das aktuelle Zwischenergebnis der Umfrage sehen können sollen, oder ob diese erst nach Ende der Umfrage veröffentlicht werden soll.

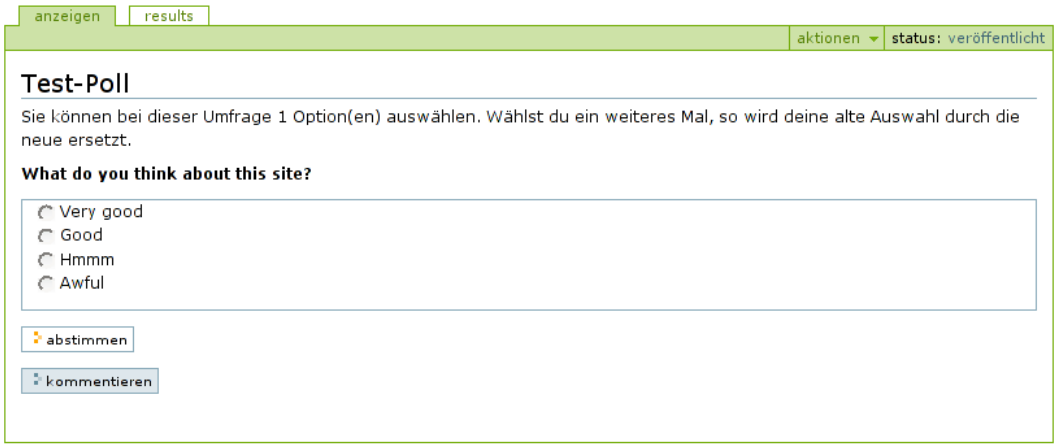

Abbildung 9: Poll - Umfrage im normalen Anzeigebereich

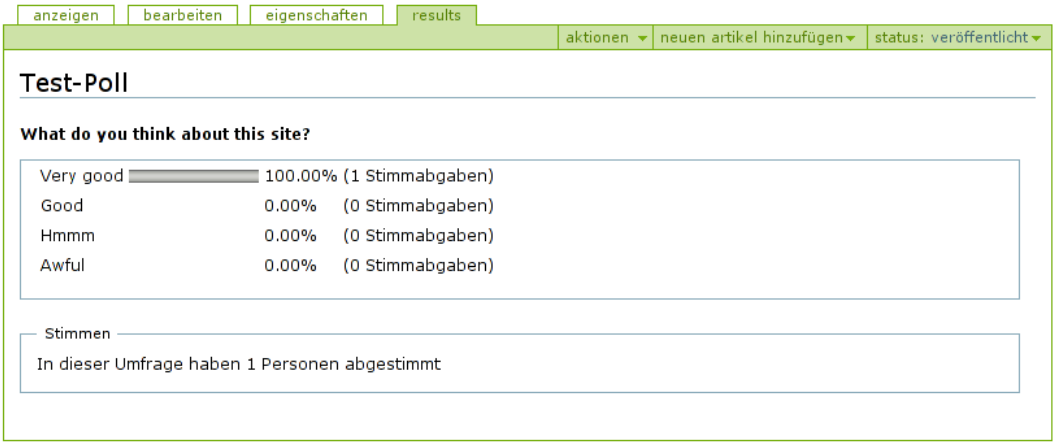

Abbildung 10: Poll - Ergebnis im normalen Anzeigebereich

Das Umfrage-Tool hat darüber hinaus den Vorteil, dass es jede Person nur einmal wählen kann. Wählt man vor Ende der Ablaufzeit ein zweites mal, so wird die ursprüngliche Auswahl geändert. Dieses Vorgehen hat nur einen kleinen „Schönheitsfehler“: wenn das Portlet-Template nicht wie oben beschrieben geändert wird und die Umfrage auch für nicht angemeldete Benutzer sichtbar ist, so können anonyme Benutzer ebenfalls wählen, allerdings werden diese Stimmen alle dem internen Benutzer „Anonymous“ zugeordnet, so dass die Wahl höchstens um eine Stimme verändert sein kann bzw. man keine „anonyme“ Umfrage starten kann. In diesem Fall könnte man im nicht angemeldeten Zustand eine Stimme abgeben und diese aus der Auswertung wieder herausrechnen.

4.8 Workflows, Sicherheit und Benutzer

Plone arbeitet zur Definition von Workflows mit DCWorkflow. Andere Systeme dafür können eingesetzt werden, in der Regel reicht jedoch die Mächtigkeit von DCWorkflow für alle Anwendungen aus.

Sämtliche Objekte in Plone haben eine Menge von Zuständen und eine Menge von Übergängen (das Konzept ähnelt dem der Endlichen Automaten). Jedes Objekt hat immer genau einen Zustand. Die Auswahl der Übergänge, die vom Benutzer ausgelöst werden können, hängen von dessen Rolle im System ab. Besser erkennt man dies in der Folgenden Darstellung des Standard-Workflows und die dazu gehörende Rechtetabelle.

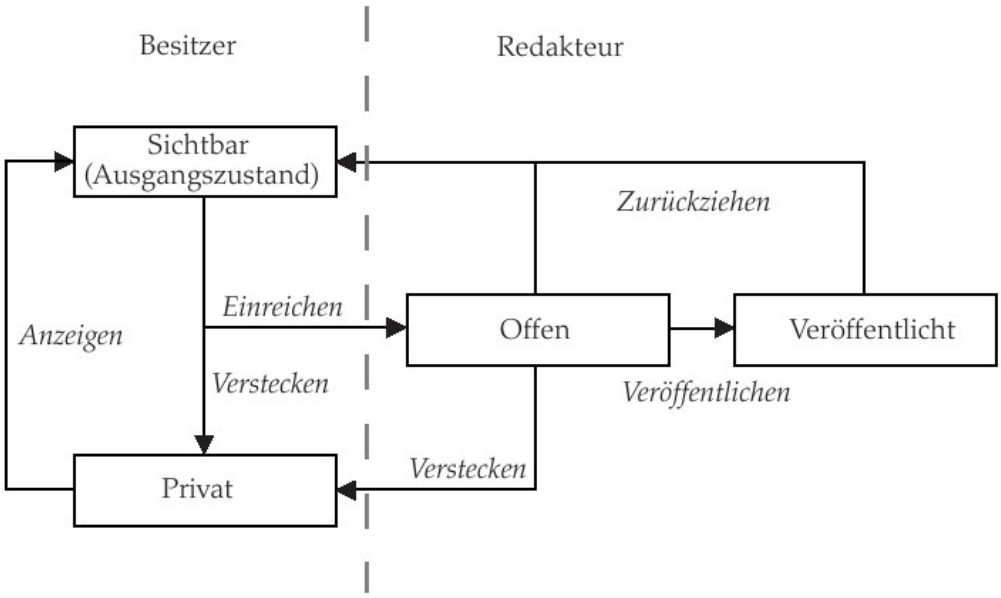

Abbildung 11: Default-Workflow in Plone für Inhalte [McKay]

Zustand	**Anonym**	**Authentifiziert**	**Besitzer**	**Manager**	**Prüfer**
Offen	Anzeigen	Anzeigen	Anzeigen	Bearbeiten	Bearbeiten
Privat	-	-	Bearbeiten	Bearbeiten	Anzeigen
Veröffentlicht	Anzeigen	Anzeigen	Anzeigen	Bearbeiten	Anzeigen
Sichtbar	Anzeigen	Anzeigen	Bearbeiten	Bearbeiten	Anzeigen

Tabelle 1: Rechte im Default Workflow [McKay]

Innerhalb von Plone können beliebig viele Workflows definiert werden und jedem Inhaltstyp kann ein eigener zugewiesen werden. Da es in der Vergangenheit schon zu „Verunstaltungen“ auf der ESSA-Seite kam, soll ein restriktiver Workflow eingeführt werden, bei dem die Anzeige der Inhalte erst von einem Manager oder Prüfer freigegeben werden soll. Der Standard-Workflow von Plone reicht für diesen Zweck schon aus, da selbst registrierte Benutzer erst einmal nur Zugriff auf Ihr eigenes Verzeichnis und das Wiki haben.

5 Implementierung des Kurskataloges (SocSimNet)

Dieser Abschnitt beschreibt die Entwicklung des Kurskataloges (SocSimNet) als Produkt in Plone.

5.1 Einleitung

Da für die Verwaltung des Kurskataloges kein passendes Produkt existiert, musste dieses komplett neu entwickelt werden. Um die Entwicklung einfach zu halten, sollte diese mit Hilfe von Archetypes und ArchGenXML erfolgen.

Der Kurskatalog soll in der Lage sein, Kurse mit Bezug zu sozialwissenschaftlicher Simulation zu verwalten. Dazu müssen die Daten der Kurse, der Veranstaltungsorte und der Lehrenden verwaltet werden.

Für jeden Kurs soll ein Titel, die Zielgruppe, eine Beschreibung, Literatur, Ziele der Veranstaltung, die Sprache(n) und eine zugehörige Internetseite gespeichert werden können. Eine Veranstaltung findet jeweils an einem Veranstaltungsort statt, der einen Titel und eine URL enthält. Sie kann von einem oder mehreren Lehrenden gehalten werden, für die jeweils der Akademische Titel, Vorname, Nachname, E-Mail-Adresse und Homepage gespeichert werden soll. Dies ist in folgendem UML-Diagramm noch einmal genau zu sehen (leider stellt Poseidon in der Community-Edition die Multiplizitäten nicht richtig dar, deshalb fehlen sie in dem Diagramm, diese sollten aber aus dem Text hervorgehen).

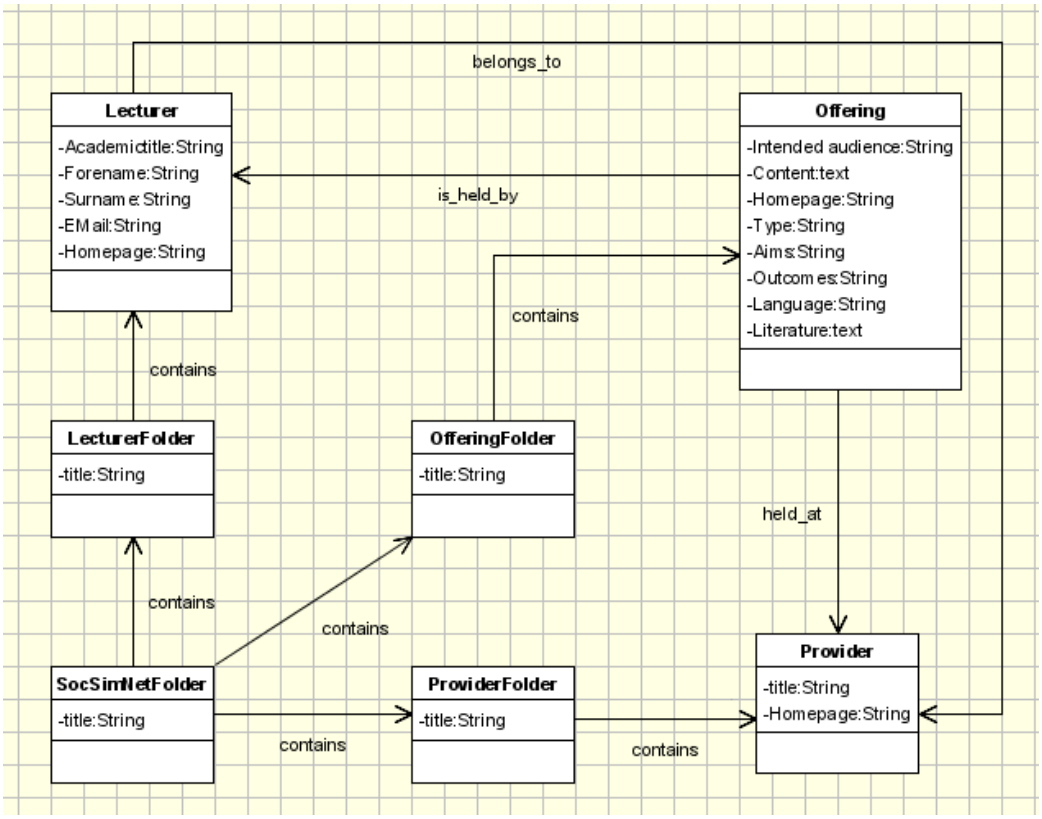

Abbildung 12: UML-Diagramm von SocSimNet

Da Plone von sich aus Referenzen auf andere Objekte verwalten kann, wäre es automatisch möglich, bei Veranstaltungen auf zugehörige Literatur aus der Bibliographie zu verweisen, ohne dies gesondert modellieren zu müssen. Trotzdem wurde später für die Literatur ein eigenes Textfeld vorgesehen, in dem diese erfasst werden kann, da dieses Feld nur als Hinweis dienen soll.

Um die Entwicklung nachvollziehen zu können, wird in den folgenden Abschnitten der Grundaufbau von Archetypes und die Funktionsweise von ArchGenXML erläutert, woraufhin Details der Implementierung genauer erläutert werden.

5.2 Archetypes

In folgenden wird eine Übersicht über die Grundlegenden Funktionen des Archetypes-Framework gegeben, mit dessen Hilfe sich neue Inhaltstypen und Produkte für Plone generieren lassen.

5.2.1 Einleitung

„Archetypes (formerly known as CMFTypes) is a framework designed to facilitate the building of applications for Plone and CMF. Its main purpose is to provide a common method for building content objects, based on schema definitions. Fields can be grouped for editing, making it very simple to create wizard-like forms.“ [1]

Diese Beschreibung von den Archetypes-Entwicklern besagt eigentlich schon, wofür Archetypes entwickelt worden ist. Es soll die Entwicklung von Produkten für Plone vereinfachen, indem es diverse Elemente und Werkzeuge bereitstellt. Es bietet unter anderem automatisch generierte Views zur Eingabe und Präsentation von Daten, Werkzeuge zur Installation und Verwaltung der damit geschriebenen Produkte und Inhaltstypen, eine Schnittstelle zur Abstraktion des Speicherortes einzelner Objektattribute, Referenzen (nur für mit Archetypes entwickelten Objekten), Standardsicherheitseinstellungen, automatische Umwandlung von verschiedenen Inhaltstypen (html, plaintext, ...) und vieles mehr.

Um mit Archetypes ein Produkt entwerfen zu können, muss erst einmal im nächsten Abschnitt der grundsätzliche Aufbau eines Produktes erläutert werden.

5.2.2 Schemata, Felder und Kontrollelemente

Wenn man einen neuen Objekttyp erzeugen will, leitet man diesen von einem Basisschema ab. Dieses Basisschema enthält zwei grundlegende Attribute, die jedes Objekt in Plone hat: der Titel und eine ID. Man erzeugt ein eigenes Schema, indem man dieses zu den Basisschema „hinzuaddiert“. Ordnerartige Objekte, welche andere Objekte enthalten können, werden statt von BaseSchema von BaseFolderSchema abgeleitet (dies wird auch in Kapitel 5.4.2 noch einmal genauer erläutert). Ein einfaches Beispiel für ein Schema mit einem Textfeld und einem Attribut ist das folgende:

```
schema = BaseSchema + Schema((
     StringField(
          'Feldname',
          required=1,
          widget = StringWidget(),
     ),
)
```

Ein Schema ist eine geordnete Python-Liste, die verschiedene Felddefinitionen enthält. Diese Felddefinitionen haben Attribute (siehe Tabelle 3), die verschiedene

Eigenschaften steuern. Jedes Feld hat ein so genanntes Widget (Kontrollelement) als Attribut, welches ein Template für die Eingabe bzw. Ausgabe bereitstellt. Der Aufbau eines Schemas geht auch noch einmal aus folgendem Bild hervor:

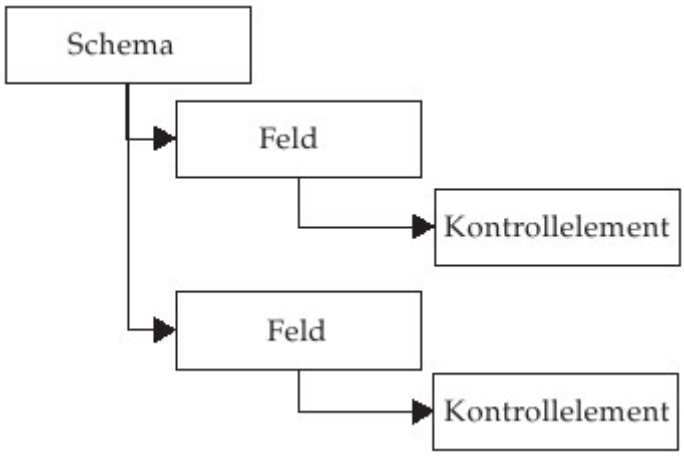

Abbildung 13: Aufbau eines Schemas [PloneBuch]

Es gibt eine ganze Reihe vorgegeber Felder (vollständige Liste siehe [2]):

Feldtyp	**Beschreibung**
CMFObjectField	Kann andere Objekte aufnehmen, und daher nur bei BaseFoldern benutzt werden
StringField	Kann Strings aufnehmen
TextField	Kann Texte aufnehmen, die auch über PortalTransforms umgewandelt werden können
IntegerField	Speichert Integer-Werte
BooleanField	Stellt einen Boolschen Wert dar, wird als Checkbox dargestellt
FileField	Kann Dateien aufnehmen
ReferenceField	Nimmt Referenzen auf andere Archetypes-Objekte auf
DateTimeField	Speichert ein Datum/Zeit-Tupel
ComputedField	Stellt ein real-only Feld dar, dessen Inhalt über eine Python-Expression berechnet wird
ImageField	Feld zum Speichern von Bildern

Tabelle 2: Verfügbare Felder [2]

Diese Felder haben verschiedene Attributen, von denen die wichtigsten in der nachfolgenden Tabelle erläutert werden sollen:

Attribut	**Beschreibung**
widget	Definiert, welches Widget dargestellt werden soll
mode	Setzt einen Zugriffsschutz für Felder, standardmäßig ist sowohl lesender als auch schreibender Zugriff erlaubt
vocabulary	Wenn gesetzt, werden vor Werte aus der angegeben Liste akzeptiert. Diese werden in der Regel als SelectionBox dargestellt
enforceVocabulary	Boolscher Wert, sorgt für doppelte Prüfung, ob ein Wert aus dem vocabulary ist
searchable	Boolscher Wert, der festlegt, ob der Feldinhalt bei einer Suche gefunden werden kann. Wenn eingeschaltet wird der Wert auch dann gefunden, wenn der Benutzer sonst kein Zugriffsrecht auf das Feld hätte
required	Boolscher Wert, ob das Feld beim Anlegen oder Editieren mit einem Wert belegt werden muss
validators	Eine Liste von Validatored (Siehe Tabelle x), die den Wert des Feldes bei Eingabe oder Änderung auf Gültigkeit prüfen
default	Standardwert für das Feld
storage	Legt fest, wo der Wert des Feldes gespeichert werden soll. Dies wird in Kapitel xxx noch genau erläutert
multiValues	Boolscher Wert, der festlegt, ob ein Feld mehrere Werte gleichzeitig besitzen kann. In Programmiersprachen mit Sets zu vergleichen. Kann nur bei Feldern wie ReferenceField oder SelectField, die mehrere Werte aufnehmen können unterstützt.

Tabelle 3: Verfügbare Attribute der Felder [2]

Für die Felder stehen diverse Widgets (siehe Tabelle x) zur Verfügung, die ihrerseits wieder verschiedene Attribute haben, welche in Auszügen in Tabelle x erläutert werden. Nicht alle Widgets lassen sich mit allen Feldern kombinieren, sinnvolle Kombinationen sollten sich aber problemlos denken lassen, ohne sie hier vollständig aufzuführen. Beispielhaft soll hier eine einfache Kombination aus String-Feld und Auswahlfeld, welches mögliche Eingaben beschränkt gezeigt werden:

```
StringField(
     'Anrede',
     vocabulary = ['Herr', 'Frau'],
     default = 'Herr',
     widget = SelectionWidget(
          label = 'Anrede',
     ),
```

Widget	**Beschreibung**
StringWidget	Eingabefeld für einfache Strings, wird als HTML-Input-Box dargestellt
TextAreaWidget	Eingabefeld für mehrzeilige Texte, wird als HTML textarea dargestellt
SelectionWidget	Stellt eine Eingabe für eine Liste von Werten dar, von denen nur einer gewählt werden darf. Je nach Länge der Liste wird es entweder als Dropdown-Feld oder als Gruppe von Radio-Boxen dargestellt
MultiSelectionWidget	Eingabefeld für Felder, die mehrere Werte aus einer Liste zulassen. Je nach Länge der Liste wird es entweder als Select-Feld oder als Gruppe von Checkboxen dargestellt.
BooleanWidget	Eingabefeld für Boolsche Werte. Wird als Checkbox dargestellt
RichWidget	Stellt eine Eingabemöglichkeit für formatierte Texte dar, in denen auch Dateien hochgeladen werden können. In der Regel wird es durch den bei Plone mitgelieferten WYSIWYG-Editor kupu dargestellt
ReferenceWidget	Feld zur Eingabe von Referenzen auf andere Objekte. Je nachdem, ob multiValues gesetzt ist oder nicht wird es als SelectionWidget oder MultiselectionWidget dargestellt
IntegerWidget	Wie StringField, nimmt aber ausschließlich Integer-Werte entgegen
CalendarWidget	Stellt eine Eingabemöglichkeit für Datum und Zeit dar, wird durch mehrere getrennte Drop-Down-Boxen dargestellt
PasswordWidget	Eingabefeld für einfache Strings, bei denen die eingegeben Werte in der Regel durch Sterne ersetzt werden. Wird nicht im view angezeigt.
ComputedWidget	Stellt den vom Feld berechneten Wert in HTML dar.

Tabelle 4: Verfügbare Widgets [3]

Attribut	**Beschreibung**
label	Beschriftung des Feldes, die in der Benutzeroberfläche erscheint
label_msgid	Beschriftungs-Id für Übersetzungsmodul
modes	Anzeigemodi des Feldes (view, edit)
description	Hilfetext, der bei der Bearbeitung des Feldes angezeigt wird
description_msgid	Hilfe-Id für Übersetzungsmodul
visible	Sichtbarkeit des Widgets. Möglich sind visible, hidden und invisible
postback	Legt fest, ob ein Feld nach Auftreten eines Fehlers beim Editieren wieder mit dem zuletzt eingegebenen Inhalt gefüllt werden soll. Dies ist ein Boolescher Wert.
populate	Legt fest, ob beim Editieren eines Feldes der Inhalt des vorhandenen Objektes übernommen werden soll. Der Wert ist ein Boolescher Wert und (außer bei Passwortfeldern) immer auf True voreingestellt.
i18n_domain	Legt eine Internationalisierungs-Domäne fest, die für das Übersetzungsmodul gebraucht wird. Hiermit ist es möglich, ein eigenes Vokabular für das Produkt zu definieren. (Wenn dort kein Wert für das Feld/Beschreibung vorhanden ist, wird versucht, das von Plone vorgegebene zu verwenden)

Tabelle 5: Verfügbare Attribute für Widgets [3]

Um die Eingaben auf Korrektkeit zu überprüfen stehen zusätzlich noch einige Validatoren zur Verfügung, die dem Feld als Attribut zugeordnet werden können:

```
validators = ("validator1", "validator2")
```

Eine Liste der wichtigsten Validatoren zeigt die folgende Tabelle:

Validator	Beschreibung
isDecimal	Überprüft, ob der eingegebene Text einer Dezimalzahl entspricht. Beinhaltet positive und negative Zahlenliterale sowie die Exponentialnotation.
isInt	Überprüft, ob der Text eine Ganzzahl ist.
isPrintable	Überprüft, ob ein Text druckbar ist, also nur aus Buchstaben, Ziffern und dem Dollarzeichen besteht.
isSSN	Prüft, ob ein Text eine gültige US-Amerikanische Sozialversicherungsnummer ist.
isUSPhoneNumber	Prüft, ob ein Text eine gültige US-Amerikanische Telefonnummer ist.
isInternational-PhoneNumber	Prüft, ob ein Text eine gültige internationale Telefonnummer ist.
isZipCode	Prüft, ob ein Text eine gültige US-Postleitzahl mit 5 oder 9 Zeichen ist.
isURL	Prüft, ob ein Text mit http://, ftp:// oder https:// beginnt.
isEmail	Prüft, ob ein Text eine gültige E-Mail-Adresse ist.

Tabelle 6: Validatoren [McKay]

Weitere Validatoren können relativ einfach selbst programmiert werden. Dazu muss man nur eine Klasse schreiben, die das Ivalidator-Interface implementiert. Zu diesem Zweck existieren auch schon zwei vorbereitete Klassen, von denen man ableiten kann: RegexValidator für Reguläre Ausdrücke und RangeValidator für Zahlenbereiche.

5.2.3 Ansichten (Views)

Archetypes kennt drei Standard-Ansichten für Objekte, die automatisch erzeugt werden: view, edit und properties. Um die automatisch generierten Ansichten anzupassen, muss man diese überschreiben. Dazu wird in der Hauptklasse des Objekts das actions-Attribut gesetzt, in diesem Beispiel wird ein anderes Page Template für die View-Action des Providers gesetzt:

```
class Provider(BaseContent):
     ...
     actions = ({
          'id': 'view',
          'name': 'View',
          'action': "string:${object_url}/provider_view",
          'permissions': ("View",),
          },)
```

Damit dieses Beispiel funktioniert, muss natürlich das Page Template „provider_view.pt“ im „skins/SocSimNet“-Verzeichnis vorhanden sein.

Darüber hinaus kann man zusätzliche Views generieren, die noch nicht standardmäßig vorhanden sind, um die Daten anders formatiert auszugeben.

5.3 ArchGenXML

Dieser Abschnitt beschreibt die Arbeit mit ArchGenXML.

5.3.1 Einführung

ArchGenXML soll das Generieren von Produkten aus UML-Diagrammen unterstützen. Dazu benötigt man im Grunde genommen nur ein UML-Modellierungstool, welches XMI-Dateien exportieren kann. Versuche hiermit wurden mit den beiden kostenlos erhältlichen Programmen ArgoUML und Poseidon (Community-Edition) unternommen.

5.3.2 Arbeitsweise und Beispiel

Zum Generieren eines Produktes wird ein Klassendiagramm als Datenmodell und eventuell ein Zustandsdiagramm für die Zustände, die die Objekte einnehmen können erstellt. Der einfachste Weg, aus diesen Diagrammen nach Export als XMI ein Produkt zu generieren, ist die XMI-Datei in das Produktverzeichnis zu kopieren und von dort aus ArchGenXML aufzurufen. In diesem Beispiel wäre dies im Verzeichnis Products/SocSimNet der Befehl:

```
python ../../ArchGenXML.py SocSimNet.xmi SocSimNet
```

Daraufhin wird ein Produkt gebaut, welches schon sämtliche Objektdateien, Verzeichnisse und Installationsroutinen enthält und direkt über die Plone-Konfigurationsseite installiert werden kann.

Damit manuelle Änderungen am Code bei einem erneuten Durchlauf nicht überschrieben werden, werden dort geschützte Bereiche erstellt, die bei einem neuen „build“ berücksichtigt werden. Diese Bereiche sind mit #code-section markiert. Zusätzlich werden die Page-Templates, die im skins-Ordner stehen nicht überschrieben.

Bei der Generierung werden dabei sehr viele UML-Konzepte unterstützt. Klassen stehen jeweils für einen Inhaltstyp, wobei auch abstrakte Klassen möglich sind (diese sind später nicht sichtbar, da sie nicht instanziiert werden können).

Methoden werden ebenfalls als Methoden des Inhaltstyps generiert. Werden Methoden mit dem Stereotyp „view“ oder „form“ ausgezeichnet, werden diese als Page-Templates generiert (diese werden aber nicht erstellt!!). Der Stereotyp „action“ einer Methode sorgt dafür, dass diese als Aktion generiert wird (z.B. „veröffentlichen“). Klassenattribute werden als Felder des Inhaltstyps generiert. Generell kann ArchGenXML mit Generalisierungsbeziehungen umgehen, Mehrfachvererbung wird hierbei ebenfalls unterstützt. Bei Verwendung von Aggregation oder Composition werden Ordnerartige Inhaltstypen generiert. Assoziationen erzeugen Referenzen der Objekte untereinander, es werden aber ausschließlich die Kardinalitäten 0..1 und 0..n unterstützt. Sämtliche zusätzlichen Einstellungen werden über Stereotypen und Tagged Values eingestellt bzw. per Kommandozeile mit an den Generator übergeben. Gerade diese Stereotypen und Tagged Values sorgen aber sehr schnell dafür, dass das Arbeiten mit den UML-Diagrammen unübersichtlich wird, da sie nicht in der normalen Ansicht der Editoren zu sehen sind, sondern erst, wenn man sich die Details eines Objektes ansieht.

Um die Funktionsweise von ArchGenXML zu testen, wurden die Grundlegenden Objekte von SocSimNet (Provider, Lecturer und Offering) als Klassen erzeugt und die Beziehungen untereinander festgelegt, wonach ein Generierungslauf gestartet wurde. Nach einigem Testen hat sich hier leider herausgestellt, dass keines der verwendeten UML-Modellierungstools in der Lage war, XMI-Dateien korrekt zu exportieren oder zu importieren. Alle Tools versagten beim Speichern oder Laden der Attribute der Klassen, so dass ein sinnvolles Arbeiten hier nicht möglich war. Trotzdem hat ArchGenXML eine ordentliche Verzeichnisstruktur inklusive Installationsskripten angelegt, so dass ein Grundgerüst zur verfügung stand, auf dem die Entwicklung dann von Hand weitergeführt wurde. Wenn dieses Produkt noch weiter verbessert wird, ein guter UML-Modellierer zur Verfügung steht und vor allen die mehr als mangelhafte Dokumentation einmal in einem Zustand ist, mit dem man sinnvoll arbeiten kann, wird dieses Tool sicher die Arbeit bei der Implementierung von neuen Produkten für Plone stark vereinfachen.

5.4 Implementierungsdetails

In diesem Abschnitt werden Teile des Quellcodes und der Aufbau des Produktes im Detail beschrieben.

5.4.1 Struktur

Verzeichnisstruktur:

```
SocSimNet
     Extensions
          Install.py
          __init__.py
     skins
          SocSimNet
               Lecturer_view.pt
               Lecturer_folder_view.pt
               Offering_view.pt
               Offering_folder_view.pt
               Provider_view.pt
               Provider_folder_view.pt
          SocSimNet_public
     Lecturer.py
     LecturerFolder.py
     Offering.py
     OfferingFolder.py
     Provider.py
     ProviderFolder.py
     SocSimNetFolder.py
     __init__.py
     config.py
     refresh.txt
     version.txt
```

Der Aufbau der Verzeichnisse ist im Grunde genommen sehr einfach und bei allen Archetypes-Produkten gleich. Im Hauptverzeichnis befinden sich die einzelnen Inhaltstypen, eine Konfigurationsdatei (config.py), in der Einstellungen vorgenommen werden, die für alle Inhaltstypen gelten, und zwei Textdateien, version.txt für die Versionsnummer (damit erkennt Plone, wann ein Produkt veraltet ist und aktualisiert werden muss) und refresh.txt zum Steuern der dynamischen Neuladefunktion eines Produktes während der Entwicklung. Die __init__.py wird von Python verwendet, um zu zeigen, dass das Verzeichnis ein Python-Modul darstellt (Zope führt diese Datei beim Importieren des Paketes aus, hier kommt also Code zum Registrieren eines Produktes an Zope hinein).

Im Extensions-Verzeichnis steht ein Installationsskript, welches für das korrekte Installieren des Produkten in Plone sorgt. Im Skins-Verzeichnis stehen die Page-Templates, die für die Objekte definiert werden können. In SocSimNet wurden ausschließlich die View-Templates verändert, um eine optisch schönere Anzeige als bei Verwendung von automatisch generierten Sichten zu bekommen und einige Felder auszublenden bzw. auf eine nicht standardkonforme Weise anzuzeigen. Dies ist insbesondere bei den Foldern nötig, da man diese ansonsten nicht alphabetisch sortieren könnte.

5.4.2 Objekte

Von den Objekten sollen hier Beispielhaft auf einen normalen Inhaltstyp und auf einen Ordnerartigen Inhaltstyp eingegangen werden, um deren genauen Aufbau zu erläutern. Als erstes soll der das Lecturer-Objekt erläutert werden, da dieses alle verwendeten Merkmale enthält, danach der Lecturer-Folder. Im folgenden Codebeispiel des Lecturers wurden einige sich wiederholende Dinge weggelassen, um die Komplexität des Beispiels zu reduzieren.

```
__author__  = '''Oliver Weichert'''
__docformat__ = 'plaintext'
```

Zuerst werden die benötigten Python-Module geladen. Dazu gehören neben den benötigten Archetypes-Grundklassen auch eine Erweiterung davon (für die E-Mail und URL-Felder), die Zugriffskontrolle, die gemeinsamen Konfigurationsdaten für alle SocSimNet-Klassen und die Klasse zur Speicherung der Daten in einer MySQL-Datenbank (Hier ist das doppelte SQL zu beachten, ein Flüchtigkeitsfehler, der sehr schnell und sehr oft passiert, ist hier einfach nur MySQLStorage zu schreiben).

```
from AccessControl import ClassSecurityInfo
from Products.Archetypes.atapi import *
from Products.ATExtensions.ateapi import *
from Products.SocSimNet.config import *
from Products.Archetypes.SQLStorage import MySQLSQLStorage
```

Als nächstes wird das Schema definiert, welches die Definitionen für die Felder, deren Eigenschaften, Widgets und die Speicherengine enthält. Die Felder (außer den Referenzen) werden alle in der MySQL-Datenbank gespeichert. Dabei ist es wichtig, dass der Tabellenname der Datenbank der gleiche wie der des Inhaltstyps ist. Die Feldnamen müssen ebenfalls übereinstimmen. Sowohl Tabellenname als auch Feldname sind dabei Case-Sensitive. Das Feld Title, welches jedes Objekt besitzt, wird weiter unten ausgeblendet, da dies automatisch berechnet werden soll; sowohl das Ausblenden als auch das Berechnen erfolgt aber mit Hilfe der Klasse und wird nicht im Schema definiert. Bei E-Mail und URL ist die Anwendung von Validatoren noch einmal gut zu sehen, das Referenzfeld legt keinen oder genau einen Provider fest, bei dem der Lecturer „angestellt“ ist. Hier ist auch die Verwendung von allowed_types einmal gezeigt, da nur Provider und keine anderen Objekte referenziert werden sollen. Referenzen werden von

Archetypes in der ZODB gespeichert, genau wie die UID und ParentID der Objekte.

```
schema=Schema((
     StringField('Academictitle',
          widget=StringWidget(
               label='Academic title',
               label_msgid='SocSimNet_label_Academictitle',
               description_msgid= \
                    'SocSimNet_help_Academictitle',
               i18n_domain='SocSimNet',
          ),
          storage=MySQLSQLStorage(),
     ),
     StringField('Forename',
          widget=StringWidget(
               label='Forename',
               ...
          ),
          required=1,
          storage=MySQLSQLStorage(),
     ),
     StringField('Surname',
          widget=StringWidget(
               ...
          ),
          required=1,
          storage=MySQLSQLStorage(),
     ),
     StringField('EMail',
          widget=StringWidget(
               ...
          ),
          validators=('isEmail',),
          storage=MySQLSQLStorage(),
     ),
     UrlField('Homepage',
          widget=UrlWidget(
               ...
          ),
          validators=('isURL',),
          storage=MySQLSQLStorage(),
     ),
     ReferenceField('provider',
          widget=ReferenceWidget(
               ...
          ),
          allowed_types=('Provider',),
          multiValued=1,
          relationship='lecturer_provider'
     ),
), )
```

Daraufhin folgt die Klassendefinition des Inhaltstyps. Die Klasse Lecturer wird von BaseContent abgeleitet und besitzt die Rechte, die Plone standardmäßig vorgibt. Es werden verschiedene Eigenschaften festgelegt, die das Verhalten des Objektes steuern. Dazu zählen verschiedene „Bezeichner“, die den Inhaltstyp von

anderen unterscheidbar machen oder das Page-Template, welches bei der View-Action gewählt werden soll. Hier könnte man auch ein Icon für den Inhaltstyp definieren, wenn man (anders als der Autor) geschickt genug ist, eines zu entwerfen.

```
class Lecturer(BaseContent):
     security = ClassSecurityInfo()
     __implements__ =
(getattr(BaseContent,'__implements__',()),)
     # This name appears in the 'add' box
     archetype_name             = 'Lecturer'
     meta_type                  = 'Lecturer'
     portal_type                = 'Lecturer'
     allowed_content_types      = []
     filter_content_types       = 0
     global_allow               = 1
     allow_discussion           = 0
#    content_icon               = 'Lecturer.gif'
     immediate_view             = 'lecturer_view'
     default_view               = 'lecturer_view'
     typeDescription            = "Lecturer"
     typeDescMsgId              = 'description_edit_lecturer'
```

Im folgenden wird das Schema initialisiert, die Formulierung mittels .copy(), die eigentlich nicht nötig ist, wird benutzt um einige Fehler der Factory-Klasse in der verwendeten Version auf dem Testsystem zu umgehen. Direkt danach werden einige Eigenschaften des Title-Feldes gesetzt, damit dieses bei der Eingabe von Daten nicht erscheint und (da es per default ein benötigtes Feld ist) mit Hilfe von required=0 keinen Fehler bei der Eingabe wirft.

```
schema = BaseSchema.copy() + schema
schema["title"].required = 0
schema["title"].widget.visible = {"edit": "invisible", \
     "view": "invisible"}
```

Um das Title-Feld berechnen zu können, wird nun die von Archetypes automatisch generierte setSurname-Methode überschrieben. Diese hat zwar nicht direkt etwas mit dem Title zu tun, ist aber oben als required definiert worden und da sich der Titel aus Academictitle, Forename und Surname zusammensetzt, ist diese Methode ideal dafür geeignet. Dabei wird der Title im Grunde genommen nur konkateniert, wobei bei Vorhandensein eines Academictitle noch ein Leerzeichen eingefügt wird.

```
def setSurname(self, value):
     self.getField('Surname').set(self, value)
     academictitle = ""
     if len(self.getAcademictitle()) > 0:
     academictitle = self.getAcademictitle() + " "
```

```
            forename = self.getForename()
            surname = value
            title = academictitle + forename + " " + surname
        return self.setTitle(title)
```

Da auch die Ausgabe über ein anderes Template erfolgen soll, muss noch definiert werden, wie dieses gefunden werden kann. Dazu überschreibt man Teile des actions-Attributes. In diesem Fall soll nur das Ausgabetemplate geändert werden. Mit dem action-Attribut wird dazu der Pfad festgelegt, mit dem das Template gefunden werden kann, die Zugriffsberechtigung soll die Standardmäßige zum Ansehen sein.

```
        actions = ({
            'id': 'view',
            'name': 'View',
            'action': "string:${object_url}/lecturer_view",
            'permissions': ("View",)
        },)
```

Am Ende wird der Inhaltstyp noch an Zope registriert, der Projektname kommt hierbei aus der Konfigurationsdatei, die oben includiert wurde.

```
registerType(Lecturer,PROJECTNAME)
```

Die Objekte Offering und Provider sind analog dazu aufgebaut.

Als nächstes soll der Lecturer-Folder erläutert werden. Hier werden zuerst ebenfalls die benötigten Klassen inkludiert und das Schema vom Archetypes-BaseFolder übernommen. Die Standard-Ansicht kann durch das definieren einer modify_fti-Methode festgelegt werden. Es wird mit Hilfe von filter_content_types und allowed_content_types festgelegt, dass nur Lecturer-Objekte in diesem Ordner angelegt werden können.

```
""" SocSimNetFolder class """
import string
from urllib import quote
from types import StringType
from DocumentTemplate import sequence
from Products.CMFCore import CMFCorePermissions
from Products.CMFCore.utils import getToolByName
from Products.Archetypes.public import BaseFolderSchema, \
    Schema
from Products.Archetypes.public import BaseFolder, \
    registerType
schema = BaseFolderSchema
def modify_fti(fti):
    """ overwrite the default immediate view """
    fti['immediate_view'] = 'folder_contents'
class LecturerFolder(BaseFolder):
```

```
        """ container for """
        archetype_name = "LecturerFolder"
        filter_content_types = 1
        allowed_content_types = ('Lecturer')
        schema = schema
registerType(LecturerFolder)
```

Die anderen Ordner-Inhaltstypen (SocSimNetFolder, OfferingFolder und ProviderFolder) sind wieder analog zu diesem Beispiel aufgebaut.

5.4.3 Views

Die Views der einzelnen Objekte wurde angepasst, um eine bessere Darstellung zu gewährleisten. Für die Page-Templates der Ordnerartigen Objekte wurde dazu allerdings das Original-Template aus Archetypes genommen und modifiziert, da diese relativ umfangreich sind und nur schwer von Grund auf neu zu erstellen. Zuerst soll der genaue Aufbau des Page Templates für das Lecturer-Objekt erläutert werden, danach werden die Modifikationen des Folder-Templates genauer erläutert.

Der Aufbau des Lecturer-Views enthält zunächst die Definition der verwendeten Namespaces. Danach folgt der öffnende Body-Tag und das Einbinden der restlichen Seiteninhaltes (Header, Navigation, ...; diese werden über den „main“-slot eingebunden). Zusätzlich werden die Templates der Aktionen eingebunden.

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en"
      lang="en"
      xmlns:tal="http://xml.zope.org/namespaces/tal"
      xmlns:metal="http://xml.zope.org/namespaces/metal"
      xmlns:i18n="http://xml.zope.org/namespaces/i18n"
      metal:use-macro="here/main_template/macros/master"
      i18n:domain="plone">
<body>
<div metal:fill-slot="main">
<div metal:use-macro= \
      "here/document_actions/macros/document_actions">
      Document actions (Print, ...)
</div>
```

Als nächstes werden die darzustellenden Felder eingebunden und formatiert. Aus den Templates kann man mit Hilfe von Tales-Ausdrücken entweder direkt auf die Daten zugreifen, oder wie am Beispiel der Referenzen zu Providern mittels Python-Ausdrücken (diese werden dann in einer „Sandbox“ ausgeführt). Zur Formatierung der Ausgabe wird gewöhnliches XHTML verwendet, welches bei

Bedarf mittels CSS weiter formatiert werden kann. In diesem Beispiel werden einfach alle Felder hintereinander aufgelistet, jeweils mit Label als „Überschrift“. Die Referenz-Felder sind dabei als Schleifen ausformuliert, da es mehrere Referenzen pro Lecturer geben kann (nicht unbedingt bei Provider, aber es wäre denkbar, dass ein Lecturer bei mehreren Providern angestellt ist). Bei diesem Beispiel fällt noch die Ermittlung der Offerings auf, da die Referenzen vom Offering auf den Lecturer zeigen und man deshalb die BackReferences holen muss.

```
<h1 tal:content="context/Title">Full name of lecturer</h1>
<br />
<label><span>E-mail</span>:</label><br />
<div class="field">
     <a href="#" tal:define="email context/getEMail" \
          tal:attributes="href string:mailto:$email">
          <span tal:content="email">EMail</span>
     </a>
</div>
<label><span>Homepage</span>:</label><br />
<div class="field">
     <a href="#" tal:define="homepage context/getHomepage" \
          tal:attributes="href homepage">
          <span tal:content="homepage">Homepage</span>
     </a>
</div>
<label><span>Provider</span>:</label><br />
<ul>
     <li tal:repeat="item python:here.getRefs(relationship= \
          'lecturer_provider')">
          <img src="#" tal:attributes="src item/getIcon" />
          <a href="#" tal:attributes="href \
               item/absolute_url">
               <span tal:content="item/title">Provider</span>
          </a>
     </li>
</ul>
<label><span>Offerings</span>:</label><br />
<ul>
<li tal:repeat="item python:here. \
     getBRefs(relationship='lecturer_offering')">
     <img src="#" tal:attributes="src item/getIcon" />
     <a href="#" tal:attributes="href item/absolute_url">
          <span tal:content="item/title">Offering</span>
     </a>
</li>
</ul>
```

Und schließlich werden die am Anfang geöffneten Tags wieder geschlossen.

```
</div>
</body>
</html>
```

Die Page-Templates der Folder wurde aus der Vorlage des BaseFolderTemplates aus der Datei folder_listing.pt des Plone-Paketes erstellt, die um nicht benötigte Formatierungen „erleichtert“ wurde. Um eine alphabetische Sortierung der Einträge zu erreichen, musste noch der Bereich tal:foldercontents angepasst werden und folgende Zeilen hinzugefügt:

```
dummy python:folderContents.sort(lambda \
     a,b:cmp(a.title_or_id(),b.title_or_id()));
folderContents folderContents| \
     python:contentsMethod(contentFilter);
```

5.4.4 Storage

Da im Hintergrund eine MySQL-Datenbank verwendet werden soll, muss diese erst einmal eingerichtet werden. Dazu wird im ZMI eine Z MySQL Database Connection hinzugefügt, in der die Verbindungsdaten eingetragen werden. Werden mehrere Datenbanken verwendet, so muss nach Installation des SocSimNet-Produktes noch über das Archetypes-Tool eingestellt werden, welche Verbindung für Lecturer, Offering und Provider verwendet werden soll.

Ein besonderes Problem war die im ZMySqlDA fehlende Unterstützung von UTF-8, welche beim Speichern von String-Feldern einen Encoding-Fehler geworfen hat, da in Python Strings standardmäßig nicht in UTF-8 codiert sind. Abhilfe konnte erst eine daraufhin angepasste Version aus dem 3rdParty-Paket der Open-Source-Version von MailManager, einem in Zope programmierten „email response management“-Tool, bringen, die unter [5] zu bekommen ist.

5.4.5 Probleme, die während der Entwicklung auftraten

Eines der größten Nachteile bei der Entwicklung mit Plone stellt der Zeitaufwand dar, der bei Änderungen des Quellcodes nötig ist. Wird etwas am Schema verändert oder z.B. neue Inhaltstypen hinzugefügt, so muss Plone neu gestartet und das Produkt mit dem QuickInstaller neu installiert werden. Wenn nur Änderungen am Skin durchgeführt werden und Plone im Debug-Modus läuft, werden diese sofort angezeigt. Das Neustarten ist jedoch ein aufwändiger Prozess, da die gesamte Plone-Instanz dabei neu kompiliert wird und dies auf dem verwendeten Testsystem mit relativ wenig Inhalt schon etwa zwei Minuten in Anspruch nimmt. Laut Dokumentation läßt sich dies vermeiden, indem im ZMI unter Control_Panel, Products, *Produktname* unter dem Reiter „refresh“ das Auto-Refresh für einzelne Produkte einschalten, wenn eine version.txt im

Hauptverzeichnis des Produkts bei jeder Änderung hochgezählt wird und passende Einstellungen in der refresh.txt vorgenommen werden. Das Auto-Refresh ließ sich jedoch auf dem Testsystem nicht zum funktionieren überreden, ganz zu schweigen davon, dass Plone im Debug-Modus circa 10 bis 20 mal langsamer läuft als normal und ein Arbeiten auf dem Testsystem (immerhin ein 1600 Mhz Pentium IV) kaum noch möglich war.

Ein noch viel gravierenderes Problem ist jedoch nicht der Neustart nach dem Ändern eines Schemas, sondern die Frage was mit schon existierenden Objekten passieren soll, da man sonst nicht mehr auf diese zugreifen kann. Im ZMI existiert zu diesem Zweck ein Update-Schema-Werkzeug, welches im archetypes-tool gefunden werden kann. Mit diesem kann man die anzupassenden Inhaltstypen wählen, woraufhin es über die Objekte iteriert und diese anzupassen versucht.

5.5 Zusammenfassung

Die Entwicklung eines eigenen Produktes in Plone bot einige Herausforderungen, die nicht immer einfach zu lösen waren. Im Endeffekt wurden alle Anforderungen an das Produkt umgesetzt. Erschwert wurde die Entwicklung nicht zuletzt durch die Notwendigkeit, sich außer mit dem Aufbau von Plone auch mit den Details des darunterliegenden Zope auseinandersetzen muss. Eine Stellenweise sehr mangelhafte Dokumentation der Grundsysteme, die oft wesentlich älter ist als die aktuellen Implementationen der jeweiligen Systeme ist auch ein störender Aspekt (Beispielhaft seien z.B. Methoden, die im Quelltext andere Namen haben als in der Dokumentation erwähnt).

Des weiteren ist es sehr schwierig, auf alle Eigenschaften von Plone mit Hilfe von Archetypes zuzugreifen, da erst in der Plone-Version 2.5 geplant ist, alle Inhaltstypen auf dieses Architektur umzustellen. Trotzdem liefert Archetypes ein schon sehr gut funktionierendes Framework, mit dem man – sobald man mit dessen Eigenschaften vertraut ist – doch relativ gut arbeiten kann.

6 Migration der Daten in das neue System

Für die Migration der Daten in das neue System wurde entschieden, dies manuell durchzuführen, da es sich auf Grund der geringen Datenmenge nicht lohnt, Import-Tools zu programmieren und eine Überarbeitung der Daten auch dann noch notwendig wäre, da die Daten nicht nach Content-Typen getrennt gespeichert waren.

Die Startseite und alle anderen festen Seiten wurden von Hand neu erfasst.

Die Newsletter wurden erst einmal eins zu eins übernommen, diese lagen schon im ursprünglichen System mit falscher Zeichensatzkodierung und fehlerhaften Steuerzeichen vor und eine komplette Überarbeitung hätte für die Studienarbeit zu viel Zeit in Anspruch genommen. Die Empfängerliste des Newsletters, die von der Gesamtbenutzerliste der ursprünglichen Seite abweicht, wird übernommen, sobald das neue System in Betrieb geht. Dazu kann die Liste (die sich leider nicht exportieren läßt) aus dem alten System in eine Tabellenkalkulation kopiert und als CVS-Datei wieder exportiert werden, welche von PloneGazette importiert werden kann (es werden nur Empfängername und E-Mail-Adresse benötigt).

Für die Benutzerdaten wurde entschieden, diese nicht in das neue System zu übertragen, statt dessen sollen die Benutzer des alten Systems sich neu registrieren, bei dieser Gelegenheit werden auch gleichzeitig die dazugehörigen Daten auf einen aktuellen Stand gebracht.

7 Ausblick

Da sich auch während der Umsetzung der Implementierung die Anforderungen an das System immer wieder geändert haben, wurde ein Schnittpunkt zum 09.11.2006 gezogen und in dieser Arbeit nur die bis dahin umgesetzten Aspekte beschrieben. Im Rahmen einer HiWi-Stelle wird das System aber auch in Zukunft weiterentwickelt und (sobald verfügbar) in einer Produktionsumgebung eingesetzt werden.

Zu den hier nicht besprochenen Anforderungen zählen zahlreiche gewünschte Änderungen an der Benutzerverwaltung, deren Implementierung zwar an sich nicht schwierig ist, aber einen erheblichen Aufwand bedeuten, da sehr viele Templates des Systems dahingehend angepasst werden müssen.

Des weiteren werden noch einige optische Verschönerungen an SocSimNet vorgenommen, bis jetzt wurden nur die Standard-Icons von Plone für Dokumente und Verzeichnisse verwendet, hier kann man sicher passendere entwerfen. Außerdem stört die Anzeige der Referenzen beim Anlegen bzw. Editieren der Objekte in SocSimNet, da nicht nur der Titel sondern auch der komplette Pfad in der Select-Box mit angezeigt wird.

Eine Migration auf ein Produktivsystem wird auch keine großen Probleme verursachen, hier reicht es, die Plone-Instanz aus dem ZMI zu exportieren und in auf einer anderen Zope-Installation zu importieren. Die MySQL-Datenbank, die für SocSimNet verwendet wird, kann ebenfalls problemlos auf einen anderen Server migriert werden. Insgesamt sind nur Änderungen an den Einstellungen der Server-Adressen von Datenbank- und Mailserver vorzunehmen und das System ist wieder einsatzbereit.

Literaturverzeichnis

[McKay] Andy McKay, Plone – Leitfaden für Administratoren und Entwickler, Addison-Wesley, 2005, München

[WerthKrämer] Bodgan Werth und Thomas Krämer, Entwurf und Implementierung eines Web-Portals für die ESSA (European Social Simulation Association), Studienarbeit an der Universität Koblenz-Landau, 2003, Koblenz

Internetquellen (Abgerufen am 30.11.2006):

[1] Hauptseite des Archetype-Frameworks
http://plone.org/products/archetypes

[2] Plone-Dokumentation der Felder
http://plone.org/products/archetypes/documentation/ \
manual/quickref/field-reference

[3] Plone-Dokumentation der Widgets
http://plone.org/products/archetypes/documentation/ \
manual/quickref/widget-reference

[4] Wikipedia
http://de.wikipedia.org/wiki/Wiki

[5] Downloadseite für das 3rd-Party-Modul von MailManager
http://sourceforge.net/project/showfiles.php?group_id=85788

[6] Wikipedia: Aufbau eines Content-Management-Systems
http://de.wikipedia.org/wiki/Bild:Schema_wcms_aufbau.png

[7] Einfache Übersicht über Plone für Anwender
http://jensquadrat.com/service/article/plone-grundriss-und-handbuch.pdf

[8] Wikipedia: Portal
http://de.wikipedia.org/wiki/Webportal

CD-Inhalt

Die beiliegende CD enthält die komplette Zope-Instanz inklusive der Datenbank. Die Daten der MySQL-Datenbank stehen im Hauptverzeichnis als SQL-Datei zur Verfügung.

Die Sourcen von SocSimNet sind in zopeinstance/Products/SocSimNet zu finden.

Mit dem Inhalt der CD läßt sich nach Installation der in Kapitel 4 beschriebenen Programme mit den entsprechenden Versionen die Produktionsversion komplett wieder herstellen, es wird nur noch eine entsprechende MySQL-Datenbank benötigt, deren Benutzerdaten über das ZMI entsprechend angepasst werden müssen.

Für die Produktionsversion sind einige Benutzer angelegt worden, wobei die wichtigsten der Zope-Administrator (zopeadmin) und für Plone der Manager (Manager) sind. Die Passwörtere lauten jeweils wie die Benutzernamen.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig verfasst, keine anderen als die angegebenen Hilfsmittel verwendet und alle Stellen, die anderen Werken dem Sinn oder Wortlaut nach entnommen sind, unter Angabe der Quelle kenntlich gemacht habe.

Vallendar, 07.12.2006 - Oliver Weichert