Verteilter Publikationsdienst für Plone-Inhalte

Jonas Baumann

9. März 2009

Auftraggeber	4teamwork GmbH
Projektname	Verteilter Publikationsdienst für Plone-Inhalte
Projektkürzel	publisher
Autor	Jonas Baumann
Version	0.8
Verteiler	☒ Jonas Baumann ☒ Pascal Habegger ☒ Antonio Di Luca ☒ Cengiz Cetinkaya
Druckdatum	9. März 2009
Status	◇ In Arbeit ◇ In Prüfung ◆ Fertig gestellt
Dokumenttyp	LaTeX

Nr.	Datum	Version	Kommentar	Autor
1	16. Februar 2009	0.1	Initialversion	Jonas Baumann
2	19. Februar 2009	0.2	Management Summary und Teile der Aufgabenstellung und des Konzepts erstellt.	Jonas Baumann
3	20. Februar 2009	0.3	Layout Anpassungen, Projektjournal hinzugefügt	Jonas Baumann
4	2. März 2009	0.4	Rechtschreibekorrektur, verschiedene Teile ergänzt	Jonas Baumann
5	5. März 2009	0.5	Umstrukturierung Testverfahren & Variantenentscheid	Jonas Baumann
6	6. März 2009	0.6	Rechtschreibung und z.T. Logik korrigiert	Jonas Baumann
7	9. März 2009	0.7	Source-Code hinzugefügt	Jonas Baumann
8	9. März 2009	0.8	Dokument abgeschlossen	Jonas Baumann

Inhaltsverzeichnis

Management Summary

Die Individuelle Praktische Arbeit (IPA) besteht aus der Implementation eines Moduls, das die Publikation von Plone Inhalten von einer Quellinstanz auf dezentrale Zielinstanzen mit eigener Datenbank ermöglicht (siehe Abbildung 1).

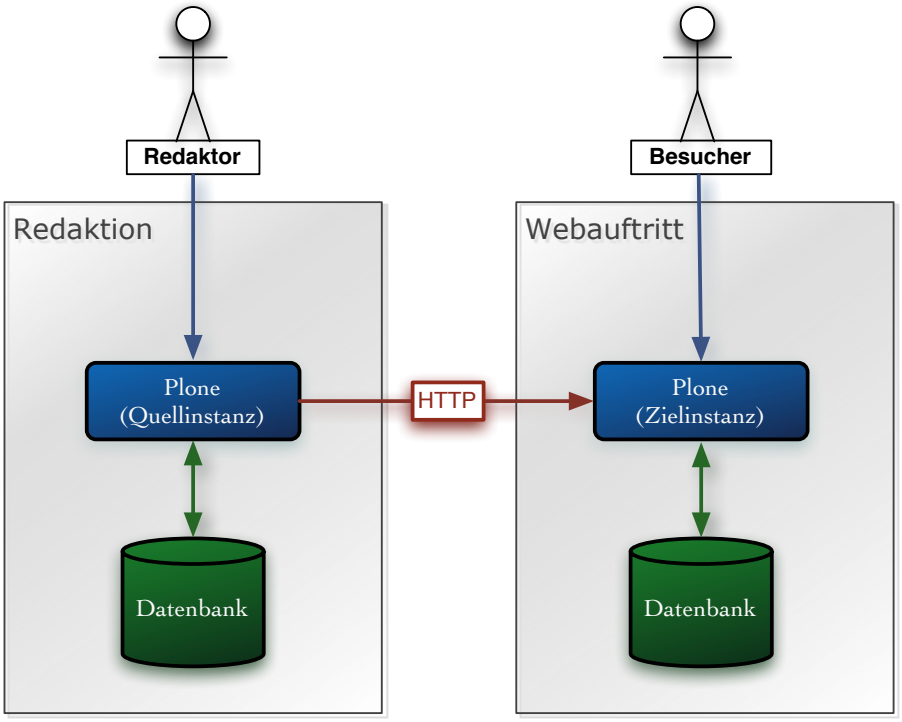

Abbildung 1: Infrastruktur

Das Produkt ist nicht für eine bestimmte Plonelösung gedacht, sondern als Komponente, die bei Bedarf installiert und konfiguriert werden kann. Für den produktiven Einsatz wird jeweils ein Integrationsmodul geschrieben, welches das projektspezifische Verhalten des Publikations-Moduls definiert.

Um die Verfügbarkeit der Quellinstanz nicht zu beeinflussen, muss der Publikationsprozess asynchron geschehen.

Um später allfällige Anpassungen und Ergänzungen zu vereinfachen, soll auf ein standardisiertes Übertragungsformat gesetzt werden, welches durch einen Variantenentscheid bestimmt wird.

Beteiligte Personen

Lernender / Autor
Jonas Baumann
Lernender Informatik
4teamwork GmbH
Engehaldenstrasse 53
3012 Bern
Telefon: +41 31 511 04 16
Mail: j.baumann@4teamwork.ch

Fachvorgesetzter
Pascal Habegger
Leiter Entwicklung
4teamwork GmbH
Engehaldenstrasse 53
3012 Bern
+41 31 511 04 12
Mail: p.habegger@4teamwork.ch

Hauptexperte
Antonio Di Luca
Die Schweizerische Post
Information Technology Services
Webergutstrasse 12
3052 Zollikofen
Telefon: +41 31 338 28 25
Mail: dilucaa@post.ch

Nebenexperte
Cengiz Cetinkaya
SBB AG
Informatik
Lindenhofstrasse 1 / Worblaufen
3000 Bern 65
Telefon: +41 51 220 82 62
Mail: cengiz.cetinkaya@sbb.ch

Teil I.

Ablauf und Umfeld

Projekt:	Verteilter Publikationsdienst für Plone-Inhalte
Autor:	Jonas Baumann
Version:	0.8

1. Aufgabenstellung

1.1. Ausgangslage

Plone[1] ist ein Content Management System (CMS), das in der objektorientierten Sprache Python[2] geschrieben wurde. Es basiert auf dem Applikationsserver Zope[3]. Zope stellt eine eigene hierarchische Objektdatenbank zur Verfügung, die Zope Object Database (ZODB[4]).

Plone wird für kleinere wie auch grössere webbasierte Auftritte verwendet. Bei grösseren Projekten macht es Sinn, den öffentlichen Webauftritt und die Redaktion in zwei Webinstanzen aufzuteilen. Durch diese Trennung kann man einerseits die Serverlast besser verteilen, andererseits kann man die Sicherheit besser gewährleisten, indem man den Redaktoren nur auf die Redaktion Zugriff gibt.

In der Standardkonfiguration von Plone ist das Content Staging nicht vorgesehen. Wenn ein Inhalt bereits publiziert wurde und redaktionell überarbeitet werden soll, werden die Änderungen sofort sichtbar. Um dies zu verhindern, kann der Inhalt zurückgezogen d.h. auf „Privat“ gesetzt werden, was bedeutet, dass der Inhalt nicht mehr öffentlich sichtbar ist.

Wenn man die Umgebung getrennt aufsetzt, also getrennte Instanzen für Redaktion und Webauftritt einsetzt, ergibt sich das nächste Problem: Wie gelangen die Inhalte, die veröffentlicht werden sollen, auf den Webauftritt? Mit dieser Aufgabe beschäftige ich mich in dieser Arbeit.

1.2. Auftrag

Ziel dieser Arbeit ist es, einen Publikationsdienst zu entwickeln, welcher es ermöglicht, Plone-Inhalte vom Redaktions-Webauftritt auf einen oder mehrere andere Webauftritte zu publizieren. Diese Dienste müssen dabei nicht auf derselben Server-Hardware laufen oder im gleichen Subnetzwerk liegen.

Kernanforderungen:

- Die Inhalte können von **einer Quell-Instanz** auf **mehrere Ziel-Instanzen** publiziert werden
- Der Vorgang soll **asynchron** erfolgen, zu diesem Zweck soll eine **Publikations-Queue** (Warteschleife) verwendet werden
- Die Vorgänge sollen sowohl auf Quell- wie auch auf Ziel-Instanz **geloggt** werden (Log-Datei)
- Zur Übertragung soll ein **standardisiertes Format** verwendet werden (JSON oder XML)
- Es sollen folgende **Grundoperationen** unterstützt werden:

[1] http://www.plone.org
[2] http://www.python.org/
[3] http://zope.org
[4] http://wiki.zope.org/ZODB/

  - *Push*: Der Inhalt wird veröffentlicht. Existiert der Inhalt bereits auf der Zielinstanz, wird er aktualisiert, sonst wird er neu erstellt.
  - *Delete*: Der Inhalt wird von der Zielinstanz entfernt.

1.3. Mittel und Methoden

Plone ist in der Programmiersprache **Python** geschrieben. Das zu erstellende Produkt muss also zwingend auch in Python geschrieben werden.

Das Produkt wird in folgender Entwicklungsumgebung entwickelt:

- Betriebssystem: Mac OS X 10.5.6
- Hardware: MacBook Pro, 2.4 GHz Intel Core 2 Duo, 2GB DDR3 RAM
- Software
  - Editor: MacVim (vim 7.2)
  - Browser: Camino 1.6.6 (Mozilla)
  - Terminal
  - Für Dokumentation: pdflatex, bibtex
  - Grafik: OmniGraffle
  - Zeitplan: Numbers (iLife)
- Sprachen: Python 2.4, XHTML
- Technologien / Standards: JSON oder XML, HTTP oder HTTPS
- Versionierung / Backup: Subversion mit täglichem Backup auf einen dezentralen Server der Firma 4teamwork GmbH
  - URL: `https://svn.4teamwork.ch/repos/external/jbaumann/`

1.4. Abgrenzung

Das Projekt konzentriert sich auf die Übertragung der Daten mittels HTTP zwischen Quell- und Ziel-Instanz. Im Rahmen der IPA werden keine konkrete Szenarien für den produktiven Betrieb erarbeitet. Aspekte der Netzwerksicherheit oder der Konfiguration auf einem produktiven Server sprengen den Rahmen und gehören nicht in diese Arbeit.

1.5. Projektorganisation

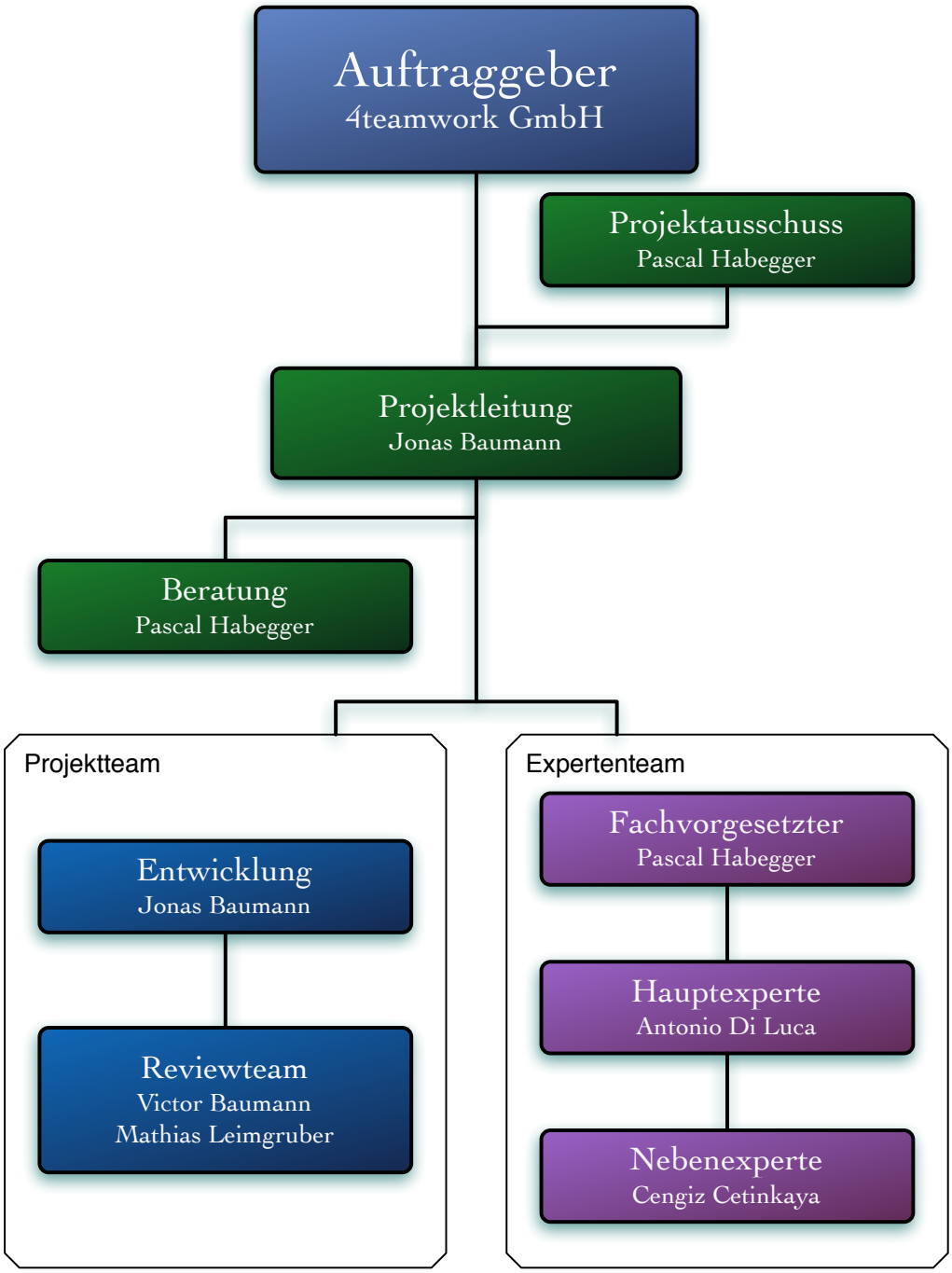

Abbildung 2: Organigramm

2. Vorkenntnisse

- Sehr gute Python- und OO-Kenntnisse (1.5 Jahre)
- Gute Kenntnisse von Plone 3.x und der Komponentenarchitektur des Anwendungsservers Zope (1.5 Jahre)
- Sehr gute Kenntnisse von XHTML, CSS, Javascript (5 Jahre)
- Vorkenntnisse im Bereich Webservices
- Vertraut mit Subversion und Plone-Buildout (1.5 Jahre)

3. Vorarbeiten

Im Rahmen der Vorarbeiten wurden zwei Produkte analysiert, die ähnliche Aufgabenbereiche abdecken: collective.synchro und ContentMirror.

3.1. collective.synchro

collective.synchro[5] wird verwendet um Inhalte zu synchronisieren. Das Produkt basiert auf ZExport, dem internen Zope Export-System. Mit ZExport können ganze Äste aus einer ZODB exportiert und in einem anderen System wieder importiert werden, sofern dieses die entsprechenden Produkte und Klassen zur Verfügung stellt.

Das Produkt unterstützt mehrere Plugins:

- **zexp**: Mit diesem Plugin werden die eigentlichen Python-Objekte mit Hilfe vom Zope internen ZExport synchronisiert. Dabei werden die Python-Objekte als Pickles[6] serialisiert.
- **fss**: Dieses Plugin wird benutzt, wenn File System Storage verwendet wird. File System Storage[7] wird gebraucht, um hochgeladene Dateien aus der ZODB auf das Dateisystem auszulagern und so die Performance zu verbessern. Dieses Plugin kümmert sich um diese Dateien und synchronisiert sie auf die Ziel-Instanz.
- **delete**: Dieses Plugin wird gebraucht um der Ziel-Instanz mitzuteilen, dass ein Objekt von der Quell-Instanz gelöscht wurde.

Der Nachteil von collective.synchro ist, dass es keine Funktion zur Übertragung der Daten an die Zielinstanz zur Verfügung stellt. Die verschiedenen Plugins generieren Dateien, die manuell auf den Zielserver kopiert werden müssen. Das Kopieren der Daten muss auf Betriebssystem-Ebene gelöst werden. Das kopieren könnte man z.B. über einen Cronjob auslösen und die Dateien über SSH oder FTP übermitteln.

3.2. ContentMirror

Das Produkt ContentMirror[8] wird für die Serialisierung von Objekten in eine relationale Datenbank (wie z.B. MySQL) gebraucht. Dies kann nützlich sein, wenn mit einer externen Applikation auf Daten zugegriffen wird, die in einer Plone-Instanz gespeichert und verwaltet werden.

Das Produkt unterstützt standardmässig alle Archetypes Inhaltstypen sowie die Standard Archetypes-Felder.

ContentMirror ist nicht geeignet für das Kopieren von Inhalten auf eine andere Plone-Instanz, da die Inhalte auf eine relationale Datenbank kopiert werden. Somit ist es für unsere Problemstellung als Lösung nicht geeignet.

3.3. Vorbereitung des Dokuments

Dieses Dokument wurde in LaTeX geschrieben. LaTeX ist ein Textsatzsystem mit eingebauter Makrosprache, basierend auf TeX. Dieses System erforder eine gewisse Einarbeitungszeit, hat aber wesentliche Vorteile in der Qualität des Layouts und verringert technische Probleme.

Ich konnte mich im letzten Jahr im Rahmen eines Projekts zur Generierung eines Buches aus Plone-Objekten mit LaTeX auseinander setzen. Mit der von mir entwickelten Applikation Skriptorium wird aus Plone-Inhalten ein LaTeX-Dokument generiert. Mit dieser Lösung wurden bereits mehrere Finanzberichte (ca. 300-500 Seiten) in Buchform gedruckt.

Die durch das Projekt Skriptorium gewonnenen Erfahrungen mit LaTeX möchte ich nun in dieser Dokumentation nutzen. So habe ich mich entschlossen meine gesamte Dokumentation in LaTeX zu „programmieren“. Dies erweitert einerseits meine Erfahrung mit dem System, andererseits erhoffe ich mir, gewisse Schwierigkeiten, die mit anderer Textverarbeitungs-Software auftreten könnten, zu ersparen.

Da diese Variante aber einen grösseren Initialaufwand für mich bedeutet, habe ich mir bereits vor Beginn der IPA die Dokumentation von der Plattform pkorg.ch heruntergeladen und diese grob in LaTeX umgesetzt. Ich habe die gesamten Überschriften in mein Dokument kopiert, die Titelseite, Fusszeile und andere layouttechnische Komponenten gestaltet und die wichtigsten Tabellen (Status- und Versionen Tabelle) vorgefertigt. So konnte ich mir mit meinem Dokument die gleiche Ausgangslage schaffen, die ich gehabt hätte, wenn ich die Microsoft Word Vorlage verwendet hätte.

[5] http://pypi.python.org/pypi/collective.synchro/
[6] http://docs.python.org/library/pickle.html
[7] http://pypi.python.org/pypi/iw.fss
[8] http://code.google.com/p/contentmirror/

4. Firmenstandards

Da die 4teamwork GmbH keine Vorlagen oder Vorgaben für Dokumente dieser Art festlegt, definiere ich das Layout und die Struktur der Dokumente individuell.

Der Python-Code wird in Anlehnung an die offiziellen Python Style Guides[9] geschrieben. Die Kommentare werden nach dem Epydoc Standard[10] gestaltet.

5. Meilensteine

Datum	Meilenstein	Erreicht
16.2.2009	Start der IPA	×
19.2.2009	Ende Analyse	×
23.2.2009	Ende Projektumriss	×
23.2.2009	Ende Konzeptionierung	×
6.3.2009	Ende Realisierung	×
9.3.2009	Projektabschluss	×
19.3.2009	Präsentation	

Tabelle 1: Meilensteine

[9] http://www.python.org/dev/peps/pep-0008/
[10] http://epydoc.sourceforge.net/epydoc.html

6. Arbeitsplan

Phase / Aktivitäten	Schätzung in h
Projekt Start	4
Dokument eröffnen	
Arbeitsplan erstellen	
Analyse	4
Prototyp mit Grobfunktionalität implementieren	
Erfahrungen auswerten und in Konzept einfliessen lassen	
Projektumriss	8
Arbeitsplan / Zeitplan integrieren / anpassen	
IST- / SOLL-Zustand analysieren	
Management Summary erstellen	
Pflichtenheft erstellen	
Konzept	6
Variantenentscheid	
Wirtschaftlichkeit / Risikoanalyse	
Spezifikationen	
Realisierung	26
Implementation Queue	
Implementation Export	
Implementation Import	
Implementation Konfiguration	
Demo-Paket / Workflow	
Fehlerbehandlung / Bugfixes	
Inline-Dokuentation	
Tests	4
Automatisierte Tests	
Testfälle spezifizieren	
Testfälle durchführen	
Dokumentation	18
Verschiedene Arbeiten am Dokument	
Diagramme erstellen	
Dokument abschliessen	
Dokument drucken und binden	
Reserve, Meetings & Verschiedenes	10
Layout Dokument (LaTeX)	
Pufferzone für ausserordentliche Probleme	
Inbegriffen Meetings / Sitzungen / Besuche	
Tagesjournal ausfüllen	
Total	**80**

Tabelle 2: Arbeitsplan

7. Zeitplan

	SOLL	IST	1	2	3	4	5	6	7	8	9	10
	in Stunden		16-Feb	19-Feb	20-Feb	23-Feb	26-Feb	27-Feb	2-Mrz	5-Mrz	6-Mrz	9-Mrz
Projekt Start	**4**	**3.5**										
Dokument eröffnen		1	1									
Arbeitsplan erstellen		2.5		2.5								
Analyse	**4**	**4.5**										
Prototyp erstellen		4.5	4.5									
Erfahrungen auswerten		0										
Projektumriss	**8**	**9**										
Arbeitsplan / Zeitplan / Meilensteine		1.5			1.5							
IST / SOLL analysieren		2		2								
Management Summary		3	2	1								
Pflichtenheft		2.5		1		1.5						
Konzept	**6**	**4**										
Variantenentscheid		3.5			1.5	2						
Wirtaschftlichkeit / Risikoanalyse		0.5			0.5							
Spezifikationenen		0										
Realisierung	**26**	**26.5**										
Implementation Queue		0										
Implementation Export		2.5				1.5		1				
Implementation Import		3.5				1.5	2					
Implementation Konfiguration		5					3	2				
Implementation Logging		5.25					1.75	3.5				
Demo Packet / Workflow		4.25						0.75	3.5			
Fehlerbehandlung / Bugfixes		2.5								2.5		
Inline-Dokumentation		3.25									3.25	
Kommentieren / Überarbeitung		0.25								0.25		
Tests	**4**	**3.75**										
Automatisierte Tests		1				1						
Testfälle spezifizieren		1.75							1.75			
Testfälle durchführen		1									1	
Dokumentation	**18**	**21.25**										
Arbeiten am Dokument		16.75			1.75	0.5	1.5			3.5	4.5	5
Diagramme erstellen		2.5							2.5			
Dokument abschliessen		2										2
Dokument drucken & binden		0										
Reserve, Meetings & Verschiedenes	**10**	**9.25**										
Layout Dokument (LaTeX)		2		1.5	0.5							
Erster Expertenbesuch		1.75			1.75							
Zweiter Expertenbesuch		1.75								1.75		
Tagesjournal ausfüllen		3.75	0.5	0.5	0.5	0.25	0.25	0.5	0.25	0.5	0.25	0.25
Informieren (PKORG, Bewertungskriterien, Buch)										0.25		
TOTAL	**80**	**82**	**8**	**8.5**	**8**	**8.25**	**8.5**	**7.75**	**8**	**8.75**	**9**	**7.25**

Abbildung 3: Zeitplan

8. Arbeitsjournal

Die Festlegungen dieses Dokuments gelten im Projekt **Verteilter Publikationsdienst für Plone-Inhalte**.

Gemäss Art. 5 Absatz 2 der Wegleitung über die individuelle praktische Arbeit (IPA) an Lehrabschlussprüfungen des BBT vom 27. August 2001 gilt:

> „Die zu prüfende Person führt ein Arbeitsjournal. Sie dokumentiert darin täglich das Vorgehen, den Stand der Prüfungsarbeit, sämtliche fremde Hilfestellungen und besondere Vorkommnisse wie z.B. Änderungen der Aufgabenstellung, Arbeitsunterbrüche, organisatorische Probleme, Abweichungen von der Soll-Planung."

Das Arbeitsjournal zur IPA ist zwingend zu führen und den Experten und Fachvorgesetzten vorzulegen. Das Arbeitsjournal ist täglich sinngemäss und korrekt auszufüllen. Das Arbeitsjournal dient der Nachvollziehbarkeit der von den Lernenden ausgeführten Arbeiten und wird als Teil der IPA in die Bewertung miteinbezogen.

8.1. Erster Tag: Montag, 16. Februar 2009

Tätigkeiten / Probleme	Aufwand geplant	Aufwand effektiv
Dokumentation: Initialversion erstellen	1.5h	1h
Management Summary: Aufgabenstellung formulieren	2h	2h
Buildout-Umgebung einrichten, Package Struktur erstellen	1h	0.5h
Prototyp für Analyse erstellen	1h	1.5h
Definition und Prototyp für Grundfunktionalität (extrahieren, codieren, decodieren)	2h	2.5h
Dokumentation: Journal vorbereiten und ausfüllen, Tagesrückblick	0.5h	0.5h

Reflexion

Zuerst habe ich mich heute mit der Aufgabenstellung auseinandergesetzt, indem ich sie in der Dokumentation ausformuliert habe. So konnte ich einen guten Zugang zum Problem finden und es besser verstehen.
Anschliessend begann ich die Umgebung einzurichten: ich erstellte ein Buildout und die geplanten Pakete mit Paster. Ich begann auch gleich damit, einige Grundfunktionalitäten gemäss Analyse-Phase zu implementieren. Dies hilft mir, das ganze Projekt besser einschätzen zu können und einige Erfahrungen im Voraus zu sammeln. So kann ich das Projekt besser Planen und Probleme voraussehen, die mir im weiteren Projektverlauf grössere Schwierigkeiten bereitet hätten. Ich konnte so einige Kernfunktionen bereits definieren und mir ein besseres Bild der Problemstellung verschaffen.
Rückblickend bin ich ein gutes Stück vorwärts gekommen, obwohl ich zu wenig Zeit in die Dokumentation investiert habe.

Nächste Schritte

Als nächstes werde ich an der Dokumentation weitermachen und das Management Summary so schnell wie möglich fertig stellen. Dann folgt das definitive Konzept und die detaillierte Planung der Arbeiten.

8.2. Zweiter Tag: Donnerstag, 19. Februar 2009

Tätigkeiten / Probleme	Aufwand geplant	Aufwand effektiv
Arbeitsplan fertig stellen	1h	0.5h
Meilensteine definieren, Zeitplan grafisch aufbereiten	1h	2h
Projektumriss: Analyse IST / SOLL	2h	2h
Management Summary erstellen	1h	1h
Pflichtenheft erstellen	2h	1h
Tagesjournal: Vorlage optimieren	1h	1h
Tagesjournal ausfüllen	0.5h	0.5h
Diverse optimierungen LaTeX	0h	0.5h

Reflexion

Heute habe ich mich vorwiegend mit der Dokumentation beschäftigt. Einen grossen Teil habe ich mit der IST- / SOLL-Analyse erledigt. Dazu habe ich zwei Diagramme gezeichnet. Ich musste heute viele Probleme mit LaTeX beheben, was mich Zeit kostete.
Auch den Arbeitsplan konnte ich heute morgen fertigstellen. Ich hoffe, dass ich diesen realistisch gestaltet habe.

Nächste Schritte

Morgen früh wird mich mein Hauptexperte Antonio Di Luca besuchen, und wir werden den aktuellen Stand des Projektes besprechen.
Ich habe vor, mich morgen nicht all zu lange mit der Dokumentation zu beschäftigen, sondern mit der Entwicklung einen Schritt weiter zu kommen.

8.3. Dritter Tag: Freitag, 20. Februar 2009

Tätigkeiten / Probleme	Aufwand geplant	Aufwand effektiv
Vorbereitungen fürs erste Treffen (Dokumentation drucken etc)	0.5h	0.5h
Treffen mit Herr Antonio Di Luca (Hauptexperte)	1h	1.25h
Anpassungen Dokumentation gemäss 1. Sitzung	0.5h	0.75h
Zeitplan anpassen: einzelne Aktivitäten einfügen	1h	1.5h
Projektjournal einrichten	1h	1h
Variantenentscheid schreiben	2h	1.5h
Abschnitt Source-Code einrichten, Beispiel einfügen	0.5h	0.5h
Wirtschaftlichkeit / Risikoanalyse anfangen	1h	0.5h
Arbeitsjournal ausfüllen	0.5h	0.5h

Reflexion

Heute morgen hat mich mein Hauptexperte besucht. Dazu habe ich mir die Zeit genommen, die momentane Version der Dokumentation zu drucken. Herr Di Luca hat mir empfohlen, den Zeitplan noch einmal zu überarbeiten und die einzelnen Aktivitäten aufzulisten. Ich habe mich anschliessend daran gemacht, das umzusetzen und einige andere kleinere Änderungen vorzunehmen, die wir besprochen hatten.
Ich habe heute auch damit begonnen, den Variantenentscheid zu schreiben. Um in der Dokumentation den Source-Code sauber darstellen zu können, habe ich eine entsprechendes „LaTeX-Environment“ eingerichtet.
Anschliessend habe ich mich mit der Wirtschaftlichkeit und der Risikoanalyse beschäftigt.

Nächste Schritte

Am Montag werde ich den Variantenentscheid fertigstellen und danach das Pflichtenheft genauer ausschreiben und die Kriterien definieren.

8.4. Vierter Tag: Montag, 23. Februar 2009

Tätigkeiten / Probleme	Aufwand geplant	Aufwand effektiv
Variantenentscheid fertig stellen	2h	2h
Pflichtenheft ausformulieren (Kriterien, Wartung, etc)	1h	1.5h
Implementation: Empfänger implementieren, Speichern der Daten	2h	1.5h
Implementation: Unittests	1h	1h
Implementation: Unterstützung für Properties	2h	1.5h
Dokumentation: Vorarbeiten	0.5h	0.5h
Arbeitsjournal ausfüllen	0.5h	0.25h

Reflexion

Heute habe ich zuerst den Variantenentscheid fertig gestellt. Anschlissend habe ich mich daran gesetzt, das Pflichtenheft genauer zu formulieren und die Kriterien zu definieren.
Dann begann ich damit, den Empfänger und das Speichern der Daten zu implementieren. Ich testete während des Entwickelns bereits einige Szenarien von Hand.
Während eines kurzen Gesprächs mit meinem Fachvorgesetzten besprachen wir, ob ich automatisierte Tests erstellen soll. Wir kamen zum Schluss, dass ich es versuchen soll. Das Problem der automatisierten Tests (z.B. Unittests) ist dass für das Testen des Publikationsprozesses zwei Plone-Instanzen in der Test-Umgebung nötig sind. Die Plone Testsuite bietet aber keine Möglichkeit, mit zwei Plone-Instanzen gleichzeit zu testen. In Absprache mit meinem Fachvorgesetzten beschlossen wir, die Tests auf manuelle Tests zu beschränken und keine automatisierten Tests einzusetzen.
Heute habe ich auch noch an einigen Themen der Dokumentation weiter gearbeitet. So habe ich noch die Vorarbeiten meines LaTeX-Dokuments deklariert.
Insgesamt bin ich heute ein gutes Stück vorwärts gekommen. Ich denke dass ich momentan recht gut in der Zeit bin und bin optimistisch, dass ich die Arbeit gut abschliessen kann.

Nächste Schritte

Am Donnerstag, meinem nächsten Arbeitstag, werde ich den Empfänger fertig implementieren. Anschliessend möchte ich mich mit dem Thema Logging auseinandersetzen. Geplant ist ein System, welches das Logging in die eigenen Logdateien übernimmt.

8.5. Fünfter Tag: Donnerstag, 26. Februar 2009

Tätigkeiten / Probleme	Aufwand geplant	Aufwand effektiv
Implementation: Empfänger implementieren, Speichern der Daten	2h	2h
Dokumentation: Probleme beim Realisieren	1h	1.5h
Entwicklung: Logging-System mit eigener Log-Datei	2h	1.75h
Entwicklung: Configlet mit Zielinstanzen	3h	3h
Arbeitsjournal ausfüllen	0.5h	0.25h

Reflexion

Man kann jetzt die gängigsten Objekte grundsätzlich publizieren und auch wieder von der Zielinstanz löschen.
Die bei der Entwicklung gemachten Erfahrungen und Probleme hielt ich im Kapitel *Realisierung* unter dem Abschnitt *Probleme* fest.
Heute habe ich mit der Implementation eines Logging-Systems begonnen. Das Problem war, dass es in einer eigenen Log-Datei gespeichert werden soll. Das Erstellen einer eigenen Log-Datei kann relativ einfach mit einem FileHandler des Python *logging*-Moduls gemacht werden. Dieser Handler kann direkt im Logger registriert werden. Leider brauchte ich realtiv lange, bis ich das herausgefunden und implementiert hatte.
Danach begann ich mit dem Entwickeln des Configlets, der Plone-Konfigurationsseite, auf welcher man die Zielinstanzen konfigurieren und administrative Tätigkeiten machen kann. Bei der Implementation der Verwaltung der Zielinstanzen hatte ich Probleme beim Generieren und Validieren des Formulars. Ich verwendete das Zope-Modul *z3c.form*, was sich als nicht sehr einfach entpuppte. Das „Erstellen"-Formular war zwar relativ schnell funktionstüchtig, beim „Bearbeiten"-Formular hatte ich aber mehr Probleme.

Nächste Schritte

Ich werde morgen weiter an den Konfigurationsseiten arbeiten und diese voraussichtlich fertigstellen.

8.6. Sechster Tag: Freitag, 27. Februar 2009

Tätigkeiten / Probleme	Aufwand geplant	Aufwand effektiv
Entwicklung: Konfigurationsseiten fertigstellen	2h	2h
Entwicklung: Logging-System fertigstellen	3h	3.5h
Dokumentation: Demo Paket & Workflow planen	1h	0.75h
Entwicklung: Unterstützung für Sortierreihenfolge implementieren	1h	1h
Arbeitsjournal ausfüllen	0.5h	0.5h

Reflexion

Heute morgen habe ich die Konfigurationsseiten fertiggestellt. Nun können Zielinstanzen hinzugefügt, bearbeitet und gelöscht werden. Die Warteschleife kann nun auch gesteurt werden (Warteschleife leeren, Warteschleife ausführen, Warteschleife anzeigen).
Dann entwickelte ich für das Logging-System die gewünschte Funktionalität. Ich baute im ganzen Produkt Logging-Befehle ein. Nun kann man alle wichtigen Informationen aus dem eigenen Log-File lesen.
Anschliessend begann ich, den Workflow für das Demo-Paket zu entwickeln. Er besteht aus drei Zuständen (Privat, Revision, Öffentlich), und daran soll dann der Publikationsdienst angehängt werden. Insgesamt bin ich heute gut vorwärts gekommen.

Nächste Schritte

Es gibt noch zwei Fehler, die ich beheben möchte:
1. Es können keine Bilder (ImageField) gelöscht werden, wenn sie bereits publiziert wurden.
2. Beim Erstellen eines Objektes wird ein „Delete"-Job erstellt.
Ansonsten werde ich mich morgen primär um die Dokumentation kümmern, diverse Diagramme (z.B. Klassendiagramme) zeichnen und aktualisieren und einige Texte dazu schreiben.

8.7. Siebter Tag: Montag, 2. März 2009

Tätigkeiten / Probleme	Aufwand geplant	Aufwand effektiv
Entwicklung: Demo Paket erstellen, Layout anpassen	1h	1h
Entwicklung: Workflow implementieren und Event-Handler erstellen	3h	2.5h
Testprotokoll erfassen	1h	1.75h
Diagramme zeichnen (Klassendiagramme)	3h	2.5h
Arbeitsjournal ausfüllen	0.25h	0.25h

Reflexion

Heute habe ich mich dem Demo-Paket gewidmet und den Workflow, den ich am Freitag spezifiziert habe, umgesetzt. Ich habe dann Event-Handler geschrieben, die bei bestimmten Transitionen (Status-Änderungen) die entsprechende Publisher-Aktion ausführen.
Das Demo-Paket beinhaltet auch einige farbliche Anpassungen, so dass man die Instanzen während der manuellen Tests und bei der Demonstration visuell besser voneinander unterscheiden kann.
Heute habe ich mich auch mit dem Testprotokoll auseinander gesetzt und geeignete Testfälle definiert.

Nächste Schritte

Am Donnerstag werde ich noch weiter an der Dokumentation arbeiten und mich auf den Besuch der Experten vorbereiten. Die beiden Fehler, die noch ungelöst sind, werde ich hoffentlich auch beheben können.

8.8. Achter Tag: Donnerstag, 5. März 2009

Tätigkeiten / Probleme	Aufwand geplant	Aufwand effektiv
Fehler: Erstellen eines Objektes macht „delete“ Job	0.5h	0.25h
Fehler: ImageField: Bild löschen funktioniert nicht	0.5h	0.25h
Fehler: Dateien (FileField) werden nicht publiziert	1h	0.5h
Fehler: Sortierung funktioniert nicht immer	1h	1h
Fehler: Objekte sollten nicht löschbar sein, wenn sie publiziert sind	0.5h	0.5h
Dokumentation: Rechtschreibeprüfung	0.5h	1h
Am Realisierungsbericht schreiben	2h	2.5h
Beurteilungskriterien überprüfen	0.25h	0.25h
Kommentieren des Source-Codes	0.25h	0.25h
Zweiter Expertenbesuch (inkl. Vorbereitung)	2h	1.75h
Arbeitsjournal Ausfüllen	0.5h	0.5h

Reflexion

Mittlerweile habe ich diverse Fehler entdeckt (siehe Journal 6. Tag), die ich noch beheben wollte. Ich habe mich daran gesetzt und eine ganze Reihe von Fehler behoben.
Diese waren z.T. noch nicht korrekt implementiert oder ich habe einen Überlegungsfehler beim Entwickeln gemacht. Das Problem, dass Objekte nicht löschbar sein sollten, wenn sie publiziert sind, betrifft den Workflow (Demo-Paket). Ich fand keine geeignete Berechtigung um das Löschen von Inhalten zu sperren: Plone bietet keine Möglichkeit, das Löschen von Objekten für berechtigte Personen zu unterbinden. Ich habe dann herausgefunden, dass das Löschen als normaler Redaktor auf einem publizierten Inhalt bereits deaktiviert ist. Lediglich der Administrator kann den Inhalt trotzdem löschen. Das ist so in Ordnung.
Löschen von Bildern: Plone hat bei Image- und File-Feldern eine spezielle Technik, um die enthaltenen Dateien zu löschen. Dies habe ich beim Entwickeln nicht beachtet. Der Mitarbeiter Mathias Leimgruber brachte mich da auf die richtige Spur und so konnte ich das Problem lösen.
Heute bin ich die Beurteilungskriterien durchgegangen. Ich muss noch einiges erledigen, z.B. den Source-Code besser kommentieren. Damit habe ich auch gleich begonnen.
Am Abend hatte ich die zweite Besprechung, diesmal mit beiden Experten (Antonio Di Luca und Cengiz Cetinkaya) und meinem Fachvorgesetzten Pascal Habegger.

Nächste Schritte

Morgen werde ich mit dem Überarbeiten des Source-Codes (Kommentieren) beginnen und wahrscheinlich einiges an der Dokumentation ergänzen und korrigieren. Ich hoffe, es reicht mir Morgen mich hinzusetzen und die ganze Dokumentation durchzulesen.

8.9. Neunter Tag: Freitag, 6. März 2009

Tätigkeiten / Probleme	Aufwand geplant	Aufwand effektiv
Kommentieren des Source-Codes	3h	3.25h
Test durchführen	1h	1h
Dokumentation: Überarbeitung / Rechtschreibeprüfung	4h	4.5h
Arbeitsjournal Ausfüllen	0.25h	0.25h

Reflexion

Als erstes habe ich heute am Kommentieren des Source-Codes gearbeitet und alle Source-Dateien mit den Kommentaren und der Kopf-Deklaration ergänzt.
Dann habe ich die definierten Testfälle durchgeführt. Der letzte Testfall war leider nicht erfolgreich, da ich beim schreiben des Tests ein Überlegungsfehler gemacht habe: Der Testfall soll sicher stellen, dass Bilder (ImageField) wieder entfernt werden können, ohne sie zu ersetzen. Dies ist beim Inhaltstyp „Bild“ nicht möglich, da dies von Plone nicht vorgesehen ist. Mit dem Inhaltstyp „Nachricht“ hat es dann wie gewünscht geklappt.
Ich habe mich dann daran gesetzt, die Dokumentation zu überarbeiten, das heisst das ganze Dokument von Anfang bis Ende durchzulesen und die Rechschreibung, den Satzbau und auch die inhaltlichen Aussagen zu korrigieren. Dies ist eine ziemlich anstrengende Aufgabe, da ich noch relativ viele Fehler im Dokument hatte. Ich konnte mich einmal grob durch das ganze Dokument arbeiten.

Nächste Schritte

Am Montag werde ich das ganze Dokument noch einmal durchlesen und sowohl inhaltliche wie auch optische Anpassungen vornehmen müssen. Danach werde ich die geforderten drei Exemplare ausdrucken und binden. Ich hoffe, dass ich auf keine grösseren Probleme stosse, denn ich muss die Dokumentation unbedingt rechtzeitig per Post versenden.

8.10. Zehnter Tag: Montag, 9. März 2009

Tätigkeiten / Probleme	Aufwand geplant	Aufwand effektiv
Dokumentation aufarbeiten, fehlende Teile erfassen	4h	4h
Source-Code in Dokumentation einfügen	1h	1h
Arbeitsjournal ausfüllen	0.25h	0.25h
Dokumentation drucken, binden und nochmals durchlesen	2h	2h

Reflexion

Heute habe ich wieder einmal von vorne mit Korrigieren begonnen. Ich hoffe, dass ich jetzt alle Fehler erwischt habe. Ich habe auch noch die fehlenden Teile ergänzt.
Dann habe ich auch noch den Source-Code im Anhang eingefügt. Ich habe nur ausgewählte Dateien eingefügt, da ich nur den zum Verständnis nötigen Quelltext drucken möchte. Die Papiermenge ist sowieso schon riesig.
Das Arbeitsjournal führe ich heute schon früh nach, da ich das Dokument nachher abschliessen möchte. Darum sind unter den oben aufgeführten Zeiten auch geschätzte.

8.11. Arbeitszeit Total

Aufwand geplant	80h
Aufwand effektiv	82h
Reflexion	Vor allem für das Dokumentieren und für das korrigieren und verbessern der Dokumentation habe ich zu viel Zeit aufgewendet. In diesem Bereich bin ich mehr als 3 Stunden über der budgetierten Zeit. Bei anderen bereichen wie Projektumriss und Konzept waren die differenzen wahrscheinlich ein Definitionsproblem. Die Zeit, die ich im einen Bereich mehr benötigte, sparte ich im anderen wieder ein. Im grossen und ganzen bin ich recht zufrieden mit den Arbeitszeiten. Aufgrund der Schwierigkeiten, die ich anfangs mit dem Zeitplan hatte, befürchtete ich grössere Differenzen.

9. Projektjournal

„Im Projektjournal werden die Informationen chronologisch gesammelt, welche im Verlauf der Arbeit eine Rolle spielten. Besprechungs-Protokolle mit Entscheide und Abmachungen sind besonders wichtig."

Quelle: Bewertungskriterium 231: Projektjournal

Datum	Ereignis / Bemerkungen
16.2.2009	**Start der IPA**
19.2.2009	**Ende Analyse**
23.2.2009	**Ende Projektumriss**
23.2.2009	**Ende Konzeptionierung**
20.2.2009	**Erster Besuch des Hauptexperten** **Themen:** • Vorstellen • Aufgabenstellung besprechen • Struktur der Dokumentation besprechen • Zeitplan besprechen **Resultate:** • Abgabe des ersten Entwurfs der Dokumention an Hauptexperten • Dokumentation: Sourcecode soll in Anhang (zählt nicht zu der vorgegebenen Seitenzahlbegrenzung) • Zeitplan: Der Zeitplan soll auch die Aktivitäten der verschiedenen Phasen enthalten • Zweiter Besuch definiert: 5. März 2009 um 16.30 Uhr
23.2.2009	**Besprechung mit Fachvorgesetzten. Thema: Unittests** **Fragestellung**: Sollen automatisierte Unittests geschrieben werden oder soll das Produkt nur manuell getestet werden? **Problematik**: Die Unittests müssen zum Testen des Publizierens auf zwei Plone-Seiten zugreifen. In Unittests von Plone steht normalerweise nur eine Plone-Instanz zur Verfügung. **Entscheidung**: Der Aufwand der Unittests soll durch das Entwickeln eines kurzen Testes gemessen werden. Bei geringem Aufwand sollen Unittests geschrieben werden, bei Misserfolg sollen ausführlichere manuelle Tests durchgeführt werden.
23.2.2009	**Entschluss Unittests** Da die automatischen Tests (Unittests) nicht mit zwei Plone-Instanzen eingerichtet werden können, werden keine automatisierten Tests geschrieben, sondern manuelle Tests durchgeführt. Details siehe Arbeitsjournal vom 23.2.2009.
5.3.2009	**Zweiter Expertenbesuch**

Datum	Ereignis / Bemerkungen
	Teilnehmer: • Jonas Baumann • Pascal Habegger • Antonio Di Luca • Cengiz Cetinkaya **Themen**: • Probleme während der IPA • Weiteres Vorgehen (Abgabe, Präsentation) • Dokumentation besprechen
6.3.2009	**Ende Realisierung**
9.3.2009	**Projektabschluss**

10. Schlussbericht

10.1. Vergleich IST / SOLL

Die Kriterien aus der Aufgabenstellung wurden erfolgreich erreicht:

- Das Produkt unterstützt eine beliebige Anzahl Ziel-Instanzen, die über die Konfigurationsseiten registriert werden können.
- Eine Warteschleife (Queue) wird für die Publikation eingesetzt. So können die Daten asynchron übertragen werden.
- Die erforderten Grundoperationen (PUSH und DELETE) wurden implementiert und funktionieren wie gewünscht
- Als Technologie wurde JSON eingesetzt, was sich bis jetzt gut bewährt hat.
- Das Logging in eine eigene Logdatei wurde implementiert und informiert über den Erfolg der Übertragung. Es wird sowohl auf der Quell- wie auch auf den Ziel-Instanzen eine Logdatei geführt. Fehler (Exceptions), die während des publizierens auftreten können, werden detailliert in die Log-Datei eingetragen.

10.2. Persönliches Fazit

Insgesamt bin ich mit dem Erfolg der Arbeit sehr zufrieden. Die Implementation geschah sogar noch schneller als erwartet, obwohl einige komplizierte Teile entwickelt wurden.

Das Produkt ist als erste Version direkt einsetzbar, obwohl noch Einiges erweitert werden könnte. Wenn die Geschäftsleitung der 4teamwork GmbH es erlaubt, werde ich das Produkt erweitern und hoffentlich unter einer OpenSource-Lizenz veröffentlichen können.

Momentan sehe ich einige Erweiterungen, die noch implementiert werden könnten:

- Versionsvergleich zwischen den Instanzen: So könnte der Redaktor sehen, was auf den Ziel-Instanzen publiziert ist und die beiden Versionen direkt vergleichen.

- „History“: Pro Inhalt können alle Publikations-Operationen und ihren Erfolg betrachtet werden.
- Betrachtung der Log-Datei über die Konfigurationsseiten.
- Bessere Unterstützung für das Verschieben von Inhalten. Momentan können die Inhalte nur verschoben werden, wenn sie nicht veröffentlicht sind.

11. Unterschriften

Die Lernende Person bestätigt mit ihrer Unterschrift diese IPA aus Eigenleistung erbracht und nach den Vorgaben der Prüfungskommission Informatik Kanton Bern erstellt zu haben. Die Angaben im Arbeitsjournal entsprechen dem geleisteten Arbeitsaufwand.

Datum	Name	Unterschrift
	Jonas Baumann , Lernender	
	Pascal Habegger , Fachvorgesetzter	

Teil II.

Projektdokumentation

Projekt:	Verteilter Publikationsdienst für Plone-Inhalte
Autor:	Jonas Baumann
Version:	0.8

12. Projektumriss

12.1. Analyse IST-Zustand

Momentan werden die meisten Auftritte und Webapplikationen mit nur einer Datenbank (ZODB) betrieben. Oft befindet sich diese auf einer Zope-Instanz, die sowohl für den öffentlichen wie auch für den redaktionellen Webauftritt verwendet wird. Bei grösseren Auftritten wird oft eine so genannte ZEO-Umgebung eingerichtet (siehe Diagramm). Mit dieser Umgebung wird die Logik von der Datenbank getrennt. So hat man einen einzelnen Datenbank-Server und eine oder mehrere ZEO-Instanzen, die auf die eine Datenbank zugreifen.

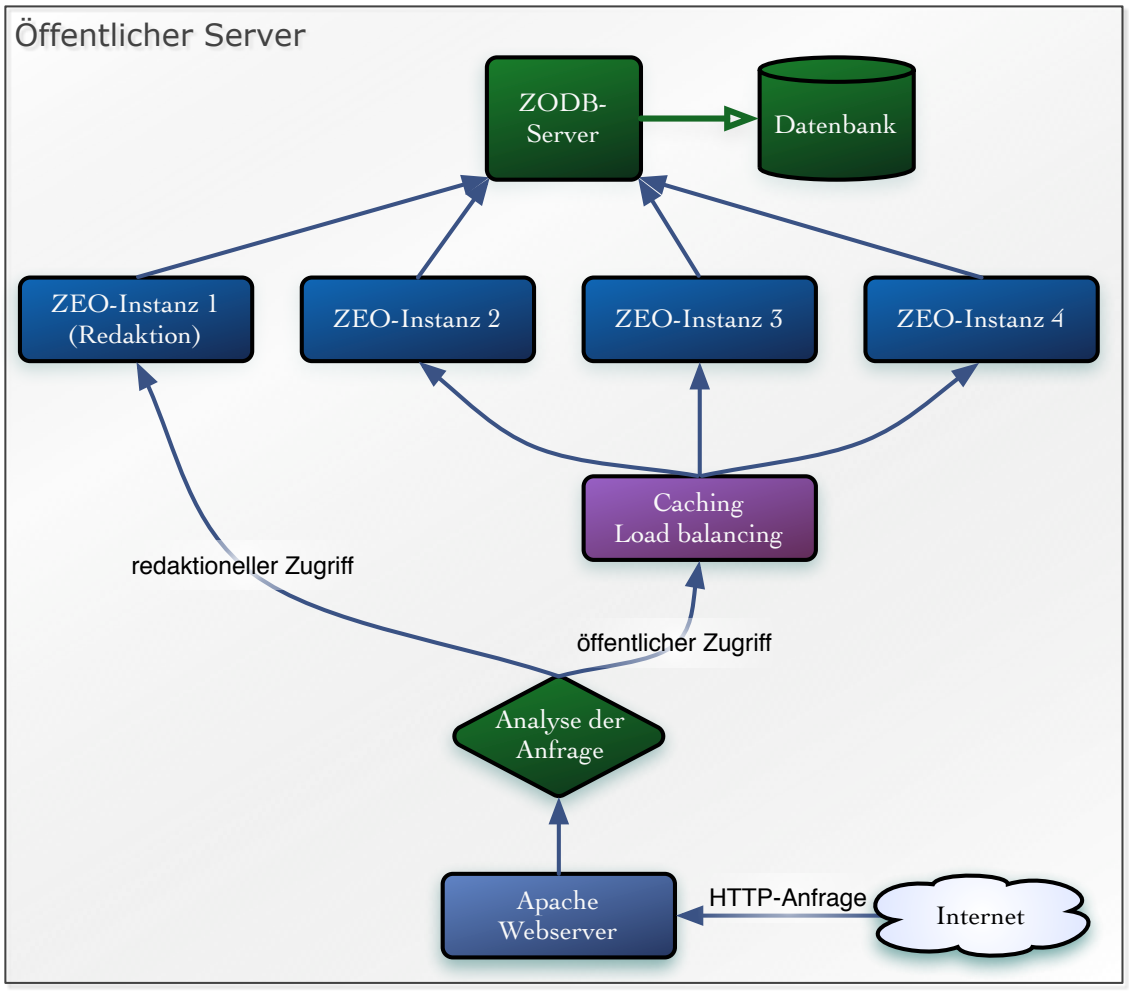

Abbildung 4: Beispiel ZEO Cluster-Umgebung

Mit dieser ZEO-Umgebung können nun verschiedene Probleme entschärft werden, aber es gibt immer noch einige Probleme.

12.1.1. Last

Die Serverlast beeinflusst sowohl Redaktion wie auch den öffentlichen Teil des Webauftrittes. Wenn zum Beispiel ausserordentlich viele Besucher gleichzeitig auf die Website zugreifen, sind die Redaktoren direkt betroffen und können möglicherweise nicht mehr richtig arbeiten. Bei speziellen Anlässen ist es durchaus denkbar, dass sich die Besucherzahl schlagartig so vervielfacht, dass die Infrastruktur, die für den normalen Betrieb ausreicht, nicht mehr genügt.

Um solchen Problemen vorzubeugen werden oft mehrere öffentliche Instanzen sowie ein Caching-System verwendet, um die Last verteilen zu können. Dadurch verschiebt sich aber der Flaschenhals nur an einen anderen Ort, denn die Instanzen müssen auf die selbe Datenbank zugreifen (siehe Abbildung 4: Beispiel ZEO Cluster-Umgebung).

12.1.2. Verfügbarkeit

Bei grösseren wie auch bei kleineren System stehen regelmässig Wartungsarbeiten an. Diese werden oft zu Randzeiten erledigt, damit die Redaktoren während der regulären Arbeitszeit in ihrer Arbeit nicht eingeschränkt werden. Bei kleineren Wartungsarbeiten reicht es oft aus, das System neu zu starten, was normalerweise höchstens wenige Minuten dauert. Wird aber ein grösseres Update eingespielt, müssen oft auch die Daten manipuliert bzw. aktualisiert werden (z.B. muss der Such-Katalog neu gerechnet werden). Solche Aktualisierungen können das System für einige Stunden sperren.

Bei grösseren, öffentlichen Webauftritten ist ein Ausfall von einigen Stunden nicht denkbar und inakzeptabel.

12.1.3. Netzwerk Sicherheit

Gerade bei grösseren Organisationen und öffentlichen Verwaltungen steht oft die Anforderung im Raum, dass die Redaktions-Instanz nicht öffentlich erreichbar ist. Es muss also möglich sein, die verschiedenen Instanzen individuell auf Netzwerk Ebene zu sichern.

Das Problem bei dieser Anforderung ist das dezentrale Betreiben der redaktionellen und der öffentlichen Instanzen auf verschiedenen Servern in verschiedenen Netzen.

Theoretisch könnte man die ZEO-Instanzen und den ZODB-Server auf mehreren Rechnern betreiben. Da entstehen aber zwei Probleme:

1. Die Kommunikation zwischen ZEO-Instanz und ZODB-Server ist über ein Netzwerk viel langsamer, als wenn sich beides auf dem gleichen Rechner befindet.
2. Der ZODB-Server (d.h. auch die Daten) befindet sich ausserhalb der geschützten Zone.

12.1.4. Content Staging

Wenn ein Redaktor einen neuen Inhalt erstellt, ist er nicht sofort öffentlich sichtbar. Sobald der Inhalt publiziert wurde und öffentlich sichtbar ist, ist es nicht immer sinnvoll, den Inhalt zu ändern. Denn jede Änderung wird sofort öffentlich sichtbar.

Um den Inhalt korrekt zu verändern muss er also wieder zurückgezogen werden und kann dann im Status „Privat“ bearbeitet werden. Das Problem ist aber, dass er dann nicht mehr öffentlich sichtbar ist und dies oft gewünscht wird.

Um dieses Problem zu lösen müsste man eine Art Kopie anlegen, die sich aber am gleichen Ort befindet. Die eine Version wäre öffentlich, die andere privat und könnte redaktionell geändert werden.

12.2. SOLL-Zustand

Für die oben beschriebene Problemsammlung gibt es eine gemeinsame Lösung: Man trennt die Datenbank in zwei Teile und richtet eine separate Datenbank für den öffentlichen und eine andere für den redaktionellen Teil des Auftritts ein.

Das Hauptproblem, das dadurch entsteht: Die Objekte müssen irgendwie von der Datenbank der Redaktion in die öffentliche Datenbank gelangen. Die Aufgabe dieser IPA-Arbeit ist es, dieses Problem zu lösen und ein Dienst zu implementieren, der die Objekte publiziert.

Eine solche Infrastruktur sieht wie in Abbildung 5 abgebildet aus.

Diese Infrastruktur hätte unten beschriebenen Einflüsse auf die genannten Probleme.

12.2.1. Last

Wie aus dem Diagramm ersichtlich kann man das Problem der Last nun individuell angehen. Eine starke Auslastung der öffentlichen Instanzen hat nun keine Auswirkungen auf die Redaktion mehr, da sich diese auf einem anderen Server befindet und unter einer anderen Domain erreichbar ist.

Trotzdem können bei grosser Besucherzahl (Hits) wieder Performance-Engpässe auftreten. Diese Engpässe können aber leichter eingegrenzt werden, indem man die öffentliche Installation komplett auf mehrere Server spiegelt.

12.2.2. Verfügbarkeit

Wartungsarbeiten auf der Redaktions-Instanz wird nun der Besucher auf einer öffentlichen Instanz nicht mehr bemerken. Für extrem lange dauernde Updates ist es sogar denkbar, dass man die Updates auf einem Testsystem durchführt und dann auf dem Produktiv-System lediglich die Datenbank austauscht.

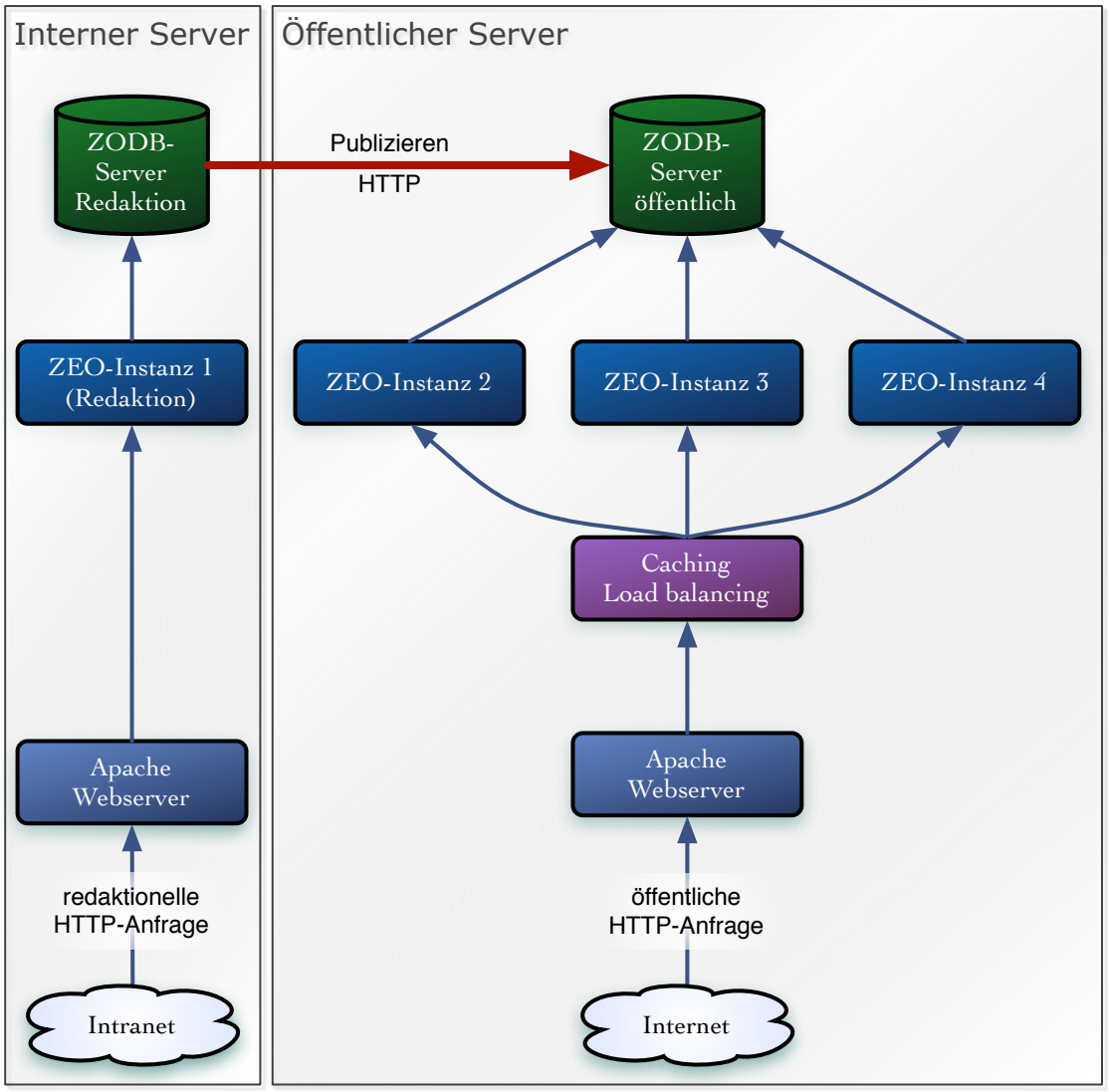

Abbildung 5: Soll-Infrastruktur mit Publikationsdienst

12.2.3. Netzwerk Sicherheit

Die Redaktion und der Webauftritt können nun in verschiedenen Sicherheitszonen betrieben werden. Dadurch kann die erwünschte Sicherheit gewährleistet werden.

12.2.4. Content Staging

Auf dem Webauftritt kann der Inhalt jederzeit von den Besuchern in der korrekten Version betrachtet werden. Gleichzeitig kann der Redaktor den Inhalt in der Redaktion anpassen und wieder publizieren. Das Content Staging ist also gegeben durch die neue Infrastruktur.

12.3. Pflichtenheft

Aus der IST- / SOLL-Analyse ergeben sich verschiedene Bedingungen für einen Publikationsdienst. Diese sind hier nochmals erläutert:

Eine Quell-Instanz, mehrere Ziel-Instanzen Es ist notwendig, dass die Redaktoren alle auf derselben Instanz arbeiten bzw. in die gleiche Datenbank speichern. Die Anzahl Ziel-Instanzen ist aber mit der in der Soll-Analyse beschriebenen Infrastruktur frei, da auf den öffentlichen Instanzen grundsätzlich keine Daten verändert werden.

Queue Eine Queue (Warteschleife) ist unbedingt notwendig, um den Publikationsprozess asynchron betreiben zu können. Der asynchrone Betrieb ist notwendig, weil nicht immer alle öffentlichen Instanzen erreichbar sein müssen (z.B. Wartung) und so das Publizieren später erfolgen kann. Der Redaktor wird so nicht aufgehalten, und es spielt nicht mehr eine grosse Rolle, ob das Publizieren schnell passiert oder ein bisschen länger dauert.

Operationen Um die Grundfunktionalität zu gewährleisten, sollen die Operationen PUSH und DELETE implementiert werden.

Die PUSH Operation wird verwendet, um einen Inhalt zu publizieren. Die Operation kann sowohl für das erstellen eines neuen Inhalts auf der Ziel-Instanz sowie für das Aktualisieren eines bereits bestehenden Inhalts gebraucht werden.

Die DELETE Operation löscht ein bereits bestehender Inhalt aus der Ziel-Instanz.

Technologie Als Technologie zur Übertragung der Daten kann XML oder JSON verwendet werden. Die genaueren Details dazu sind dem Variantenentscheid zu entnehmen.

Logging Es soll ein Logging-System implementiert werden, welches eine eigene Log-Datei für den Publikationsdienst erstellt. Die Log-Datei soll folgende Informationen enthalten:

- Datum des Ereignisses
- Ort des Ereignisses (Quell- oder Ziel-Instanz)
- Art des Ereignisses (Information, Fehler)
- Individuelle Nachricht

12.3.1. Zeitrahmen

Das Projekt soll im Rahmen der IPA realisiert werden. Der Zeitrahmen der dafür zur Verfügung steht, beträgt ungefähr 80 Stunden verteilt auf 10 Tagen. Aufgrund der Unterbrüche durch die Schule können maximal drei Tage pro Woche für das Projekt investiert werden. Das Projekt wird an folgenden Tagen durchgeführt:

- Erster Tag: Montag, 16. Februar 2009
- Zweiter Tag: Donnerstag, 19. Februar 2009
- Dritter Tag: Freitag, 20. Februar 2009
- Vierter Tag: Montag, 23. Februar 2009
- Fünfter Tag: Donnerstag, 26. Februar 2009
- Sechster Tag: Freitag, 27. Februar 2009
- Siebter Tag: Montag, 2. März 2009
- Achter Tag: Donnerstag, 5. März 2009
- Neunter Tag: Freitag, 6. März 2009
- Zehnter Tag: Montag, 9. März 2009

12.3.2. Kriterien

- Unterstützung mehrerer Zielsysteme
- Asynchrone Übertragung: Es muss eine Warteschleife (Queue) erstellt werden
- Es müssen mindestens die Aktionen PUSH (Erstellen / Aktualisieren) und DELETE (Löschen) implementiert werden
- Unterstützung für „Properties“
- Unterstützung für Ordner-Sortierung
- Datenformat: JSON oder XML
- Logging auf allen beteiligten Systemen
- Logging in eine separate Log-Datei
- Verwaltung der Zielsysteme über webbasierte Konfigurationsseiten

12.3.3. Abnahme

Nach dem Abschluss der Arbeiten und dem Durchführen der definierten Tests wird das Produkt vom Auftraggeber Pascal Habegger (4teamwork GmbH) abgenommen.

12.3.4. Verantwortung

Grundsätzlich ist das Produkt im Besitz der Firma 4teamwork GmbH und liegt in deren Verantwortungsbereich. Die Zuständigkeit liegt während der Entwicklung sowie nach der Entwicklung beim Lernenden / Autor Jonas Baumann. Er ist grundsätzlich zuständig für die Integration in andere Systeme. Das Produkt sollte jedoch so aufgebaut sein, dass jederzeit ein anderer Mitarbeiter der Firma 4teamwork GmbH jegliche Integrationsarbeiten sowie Weiterentwicklungen am Produkt durchführen kann.

12.3.5. Wartung

Die Wartung muss nicht für dieses Produkt definiert werden, da es nicht eigenständig sondern nur in anderen Projekten integriert betrieben wird. Die Verantwortung für Wartung und Betrieb liegt beim jeweiligen Verantwortlichen des Projekts, in welchem das Produkt eingesetzt wird.

13. Vorstudie

13.1. Prototyp

In der Software-Entwicklung werden heute oft iterative Projekt-Methoden eingesetzt. Das heisst, der ganze Prozess (Analyse, Planung, Realisierung, etc) wird mehrmals durchlaufen und das Produkt bei jedem Durchgang optimiert.

Zu Beginn meiner Arbeit waren mir das System als Ganzes noch zu unklar. Da ich aber zu wenig Zeit hatte, den ganzen Prozess mehrmals zu durchlaufen, entschloss ich mich, einen Prototyp in minimaler Zeit zu entwickeln. So durchlief ich eigentlich nur die Realisierungsphase im ersten Durchgang, was mir aber Klarheit über die möglichen Probleme verschaffte und die Konzeptionierung erleichterte.

Dieser erste Prototyp war jedoch noch sehr fehleranfällig. Die Fehlerbehandlung war ein Problem das mir auffiel: wie können die beiden Instanzen mit einander so kommunizieren, dass der Sender über Probleme beim Empfänger korrekt informiert wird? Wie wird der Benutzer bei Problemen informiert, denn das Ganze soll ja schlussendlich asynchron laufen?

Ich denke dieser Prototyp war eine gute Sache; ich werde in Zukunft öfters so vorgehen.

13.2. Wirtschaftlichkeit

Aufwand Der Aufwand für die Herstellung des Produkts publisher beträgt mindestens 80 Stunden (inkl. Dokumentation). Der Aufwand für die Integration in ein Projekt kann je nach Kundenwünsche variieren.

Ertrag Es werden in nächster Zeit in der 4teamwork GmbH mehrere Projekte realisiert, die eine solche Lösung benötigen. Es werden auch Projekte betrieben, die auf eine solche Lösung umgestellt werden, wenn sich diese als erfolgreich auszeichnet.

Wenn kein solches Produkt entwickelt werden würde, müsste man für diese Projekte eigenständige Lösungen suchen. Dabei würde einerseits an Komfort eingebüsst werden, andererseits müsste mit erheblich höherem Aufwand gerechnet werden, eine eigene Lösung zu finden und zu implementierten.

Nutzen Da dieses Produkt so vielseitig eingesetzt werden kann und bereits für mehrere Projekte längerfristig geplant ist, bringt es sicher erheblichen Nutzen und ist unbedingt durchzuführen.

Die durch das Projekt entstandenen Gemeinkosten können auf verschiedene Projekte und Kunden verteilt werden. Dadurch verringern sich die Kosten pro Projekt, ohne dass die Lösung an Qualität verliert. So können allenfalls Offerten kostengünstig gehalten werden.

Nach Fertigstellung des Produkts wird es möglicherweise unter einer Open-Source Lizenz veröffentlicht. Diese Entscheidung wird nach Abschluss der Arbeit getroffen. Das Veröffentlichen des Produkts bringt einerseits einen Nutzen für die weltweiten Nutzer von Plone, andererseits wird dadurch die Firma 4teamwork GmbH in der Plone-Welt noch bekannter.

13.3. Risikoanalyse

Durch unvorhergesehene Probleme könnte dieses Projekt zum Scheitern gebracht werden. Dies hätte Konsequenzen für andere geplante und bereits abgeschlossene Projekte, die das Produkt brauchen könnten. So müsste mit Mehraufwand bei diesen Projekten gerechnet werden, zusätzlich zu der in dieses Projekt investierten Zeit, die dann verloren gehen würde.

Wenn das Projekt nicht ordnungsgemäss durchgeführt werden kann, ist ein negatives Ergebnis der IPA wahrscheinlich, was wiederum Konsequenzen auf den Lehrabschluss des Lernenden Jonas Baumann haben würde.

14. Konzept

14.1. Variantenentscheid Datenübertragung

Bei PUSH-Anfragen sendet die Quell-Instanz die Daten eines Inhaltes an die Ziel-Instanz. Da es nicht möglich ist, diese Inhalte als Python-Objekte zu übertragen, müssen die Daten serialisiert werden.

Plone speichert viele Informationen in einem Objekt, die nicht übertragt werden müssen, da diese auf der Ziel-Instanz wieder generiert werden. Wir wollen diese Daten nicht übertragen, sondern uns auf die effektiven Werte der Felder des Artikeltyps konzentrieren.

Um diese Daten also einfach und effizient übertragen zu können, muss ein standardisiertes Format gewählt werden. Es gibt verschiedene Formate, aufgrund der bereits erworbenen Erfahrung in der Firma kommen aber nur zwei in Frage: XML und JSON.

14.1.1. XML

+ Vielseitig einsetzbar
+ Weit verbreitet
– Overhead
– Umständlicher Zugriff auf die Daten

Listing 1: Beispiel XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<object>
    <action>push</action>
    <data>
        <title>Hello World</title>
        <text>Das ist ein Test-Objekt</text>
    </data>
</object>
```

Grösse: 187 Byte

14.1.2. JSON

+ Kompakter
+ Einfach verarbeitbar / einfacher Zugriff
+ Schnelle Deserialisierung
– Direkte Unterstützung für binäre Daten fehlt

Listing 2: Beispiel JSON

```json
{
    'action' : 'push',
    'data' : {
        'title' : 'Hello World',
        'text' : 'Das ist ein Test-Objekt',
    },
}
```

Grösse Beispiel: 126 Byte, Grösse optimiert: 87 Byte

14.1.3. Kriterien

Der negative Punkt bei JSON, dass binäre Daten nicht unterstützt werden, kann vernachlässigt werden, da die Daten mit Hex oder Base64 codiert als String übermittelt werden können.

NR	Kriterium	Fragestellung	Gewichtung
1	Einfacher Datenzugriff	Ist der Zugriff auf die Daten einfach und schnell möglich?	5
2	Kompakt	Wie gross ist die übertragene Datenmenge?	2
3	Aufwand	Kann dies Methode innerhalb der vorgegebenen Zeit realisiert werden?	4

Tabelle 14: Kriterien Variantenentscheid

14.1.4. Nutzwertanalyse

Der Nutzwert wird wie folgt berechnet: die Punktezahl einer Variante in einem Kriterium wird mit der Gewichtung des Kriteriums multipliziert. Die resultierende Zahlen in den verschiedenen Kriterien werden addiert.

Maximale Anzahl Punkte pro Kriterium: 5

NR	Gew.	XML	Total	JSON	Total	Bemerkung
1	5	2	10	5	25	JSON kann direkt in ein Python-Dictionary konvertiert werden, auf XML muss mit dem DOM-Modell zugegriffen werden.
2	2	0	0	3	6	XML: 187 Byte (siehe Beispiel) JSON: 87 Byte (siehe Beispiel)
3	4	3	12	4	16	Beide Formate bereits angewendet: Aufwand ist bei JSON durchschnittlich kleiner
Total			**22**		**47**	

Tabelle 15: Nutzwertanalyse Variantenentscheid

Gemäss Tabelle 15 ist JSON klar die zu bevorzugende Variante, aus folgenden Gründen:

1. Direktes Konvertieren in ein Python Dictonary bzw. aus einem Python-Dictionary
2. Die Datengrösse bei JSON ist kleiner (weniger Overhead)
3. JSON ist es leichter zu verwenden, was den Entwicklungsaufwand erheblich senkt

14.1.5. Anwendungsfälle

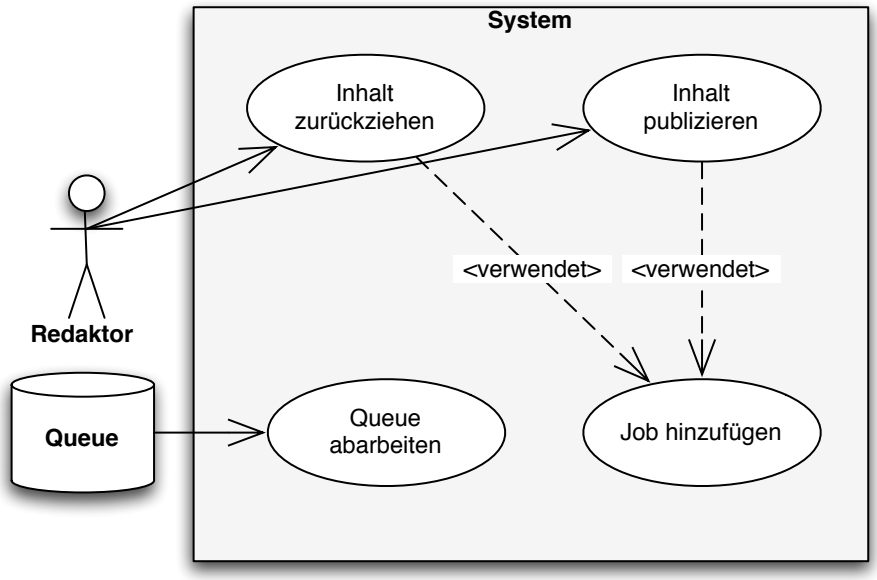

Abbildung 6: Use-Case Diagramm

Die zu entwickelnde Applikation soll die in Abbildung 6 dargestellten Anwendungsfälle implementieren:

- **Inhalt publizieren:** Ein Inhalt wird publiziert. Dies beinhaltet das Publizieren von neuen Inhalten wie auch bereits bestehenden. Es wird ein neuer Job mit der Aktion „PUSH“ in der Queue erstellt.
- **Inhalt zurückziehen:** Ein bereits publizierter Inhalt wird wieder zurückgezogen. Es wird ein Job mit der Aktion „DELETE“ erstellt.
- **Job hinzufügen:** Die beiden Anwendungsfälle „Inhalt publizieren“ und „Inhalt zurückziehen“ lösen diesen Anwendungsfall aus. Er fügt einen Job zur Queue hinzu.
- **Queue abarbeiten:** Die Warteschleife (Queue) wird abgearbeitet und die darin enthaltenen Jobs einzeln an die Ziel-Instanz gesendet.

14.1.6. Publikationsprozess

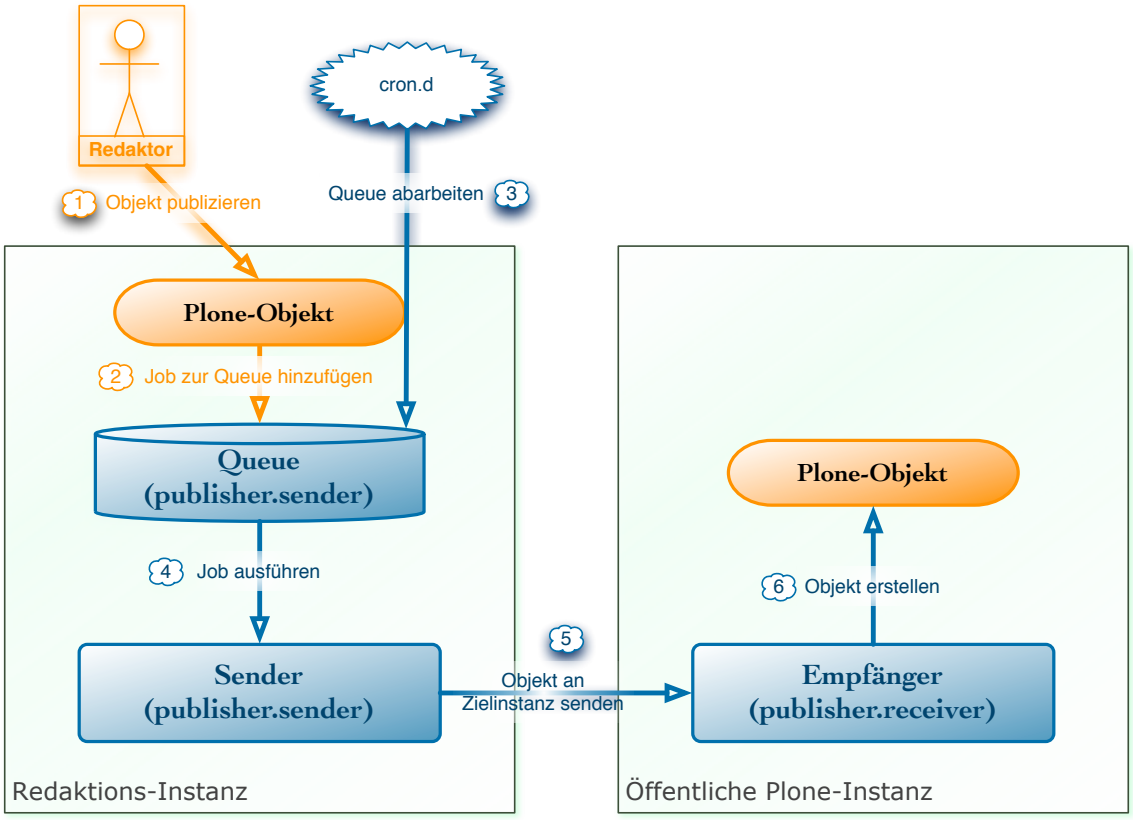

Abbildung 7: Publikationsprozess

In Abbildung 7 sind folgende Schritte illustriert:

1. Ein Redaktor löst das Publizieren eines Inhaltes aus
2. Es wird ein Job erstellt und zur Queue hinzugefügt
3. In einem bestimmten Zeitinterval wird die Queue abgearbeitet
4. Der Job wird von der Queue entfernt und abgearbeitet
5. Der Inhalt wird über eine HTTP-Anfrage an die Ziel-Instanz gesendet
6. Die Zielinstanz empfängt die Anfrage und erstellt oder aktualisiert den Inhalt

14.2. Demo Konfiguration

Um das Produkt testen und demonstrieren zu können, soll ein Demo-Paket mit einer Beispiel-Konfiguration erstellt werden. Das Paket soll folgendes abdecken:

- **Optische Unterscheidung der Quell-Instanz**: Die Quell-Instanz soll mit CSS umgefärbt werden, so dass sie gut von einer Zielinstanz unterschieden werden kann.
- **Neuer Workflow**: Um die Funktionalität einfach testen zu können soll ein neuer Workflow erstellt werden. Er soll auf das Publizieren zugeschnitten sein und Content-Staging unterstützen.
- **Publikation über Workflow**: Das Produkt soll über den Workflow angehängt werden, so dass bei den entsprechenden Transaktionen das Objekt publiziert oder zurückgezogen wird (siehe Abbildung 8).

Workflow Der Workflow besteht aus drei Zuständen:

- **Privat**: Das Objekt ist nicht publiziert und wird vom Autor bearbeitet.
- **Revision**: Das Objekt wurde bereits publiziert. Auf der Quellinstanz kann der Autor an dem Objekt arbeiten, ohne dass seine Änderungen direkt sichtbar sind (Content-Staging).
- **Publiziert**: Das Objekt wurde publiziert und kann auf der Quellinstanz nicht bearbeitet werden. Wenn am Objekt Änderungen gemacht werden sollen, muss das Objekt zuerst in den Status *Privat* oder *Revision* verschoben werden.

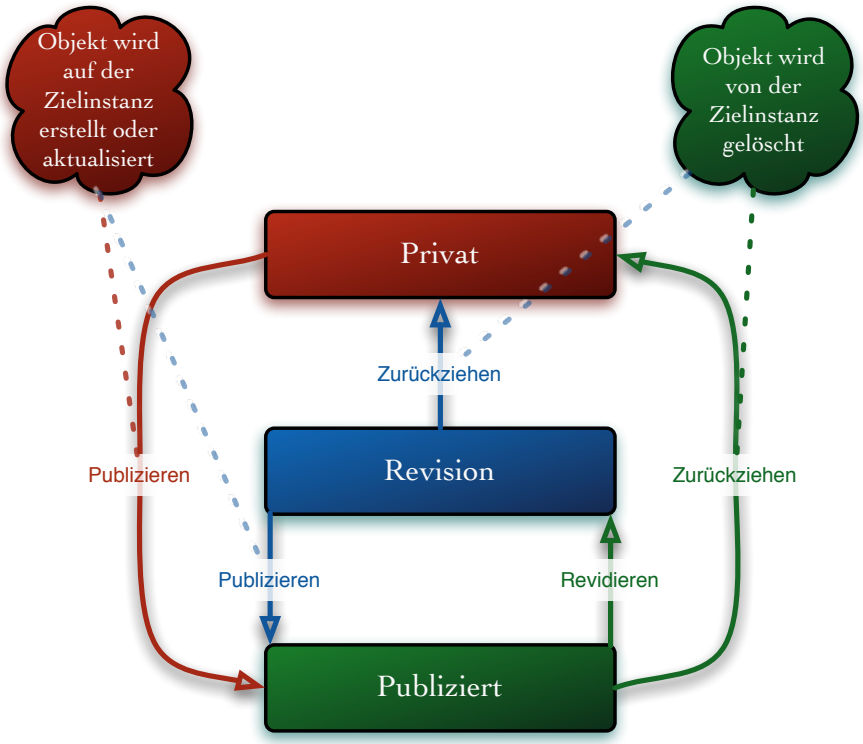

Abbildung 8: Workflow Diagramm

15. Realisierung

15.1. Klassendiagramme

Aufgrund der Aufgabenstellung ist eine Teilung in mehrere Pakete sinnvoll. So kann eine Instanz entweder als Sender (Quellinstanz) oder als Empfänger (Ziel-Instanz) eingerichtet werden. Zur Unterstützung der Kommunikation stellt ein allgemeines Paket Funktionalität zur Verfügung, die sowohl vom Empfänger wie vom Sender genutzt werden kann. Dazu gehören Klassen und Methoden zur Übermittlung des Status sowie allgemeine Funktionen.

Zusätzlich zu diesen Grundpaketen soll auch ein Demo-Paket erstellt werden. Dieses Paket enthält unterstützende Funktionalität für das Testen und für die Demonstration des Produkts.

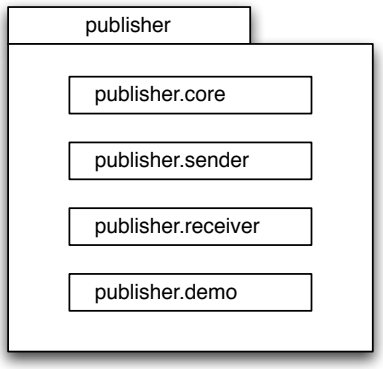

Abbildung 9: Pakete

15.1.1. Paket publisher.core

Dieses Paket muss sowohl auf Quell- wie auch auf Ziel-Instanzen installiert werden. Es enthält Status-Klassen und dazugehörige Funktionen für die Kommunikation zwischen den beiden Klassen. So kann bei einer Anfrage an die Ziel-Instanz der Status einfach an die Quell-Instanz übertragen werden, so dass auf der Quell-Instanz wieder ein Status-Objekt mit Informationen zur Verfügung steht.

Zusätzlich hat diese Lösung den Vorteil, dass man einen negativen Status (z.B. DecodeError) wie eine Ausnahme behandeln kann: mit "raise DecodeError()"wird die Transaktion mit allenfalls bereits getätigten Änderungen rückgängig gemacht, der Request abgebrochen und der Status mit der Fehlermeldung (und einem Traceback) an die Quell-Instanz zurück geschickt.

Status Klassen:

- CommunicationState
- SuccessState
- ObjectCreatedState
- ObjectUpdatedState
- ObjectDeletedState
- ErrorState
- InvalidRequestError
- DecodeError
- PartialError
- UnexpectedError

- ObjectNotFoundError

Das Paket stellt des Weiteren stellt Funktionen für das Erstellen von Logger-Objekten zur Verfügung. So kann ein einheitliches Logging gewährleistet werden.

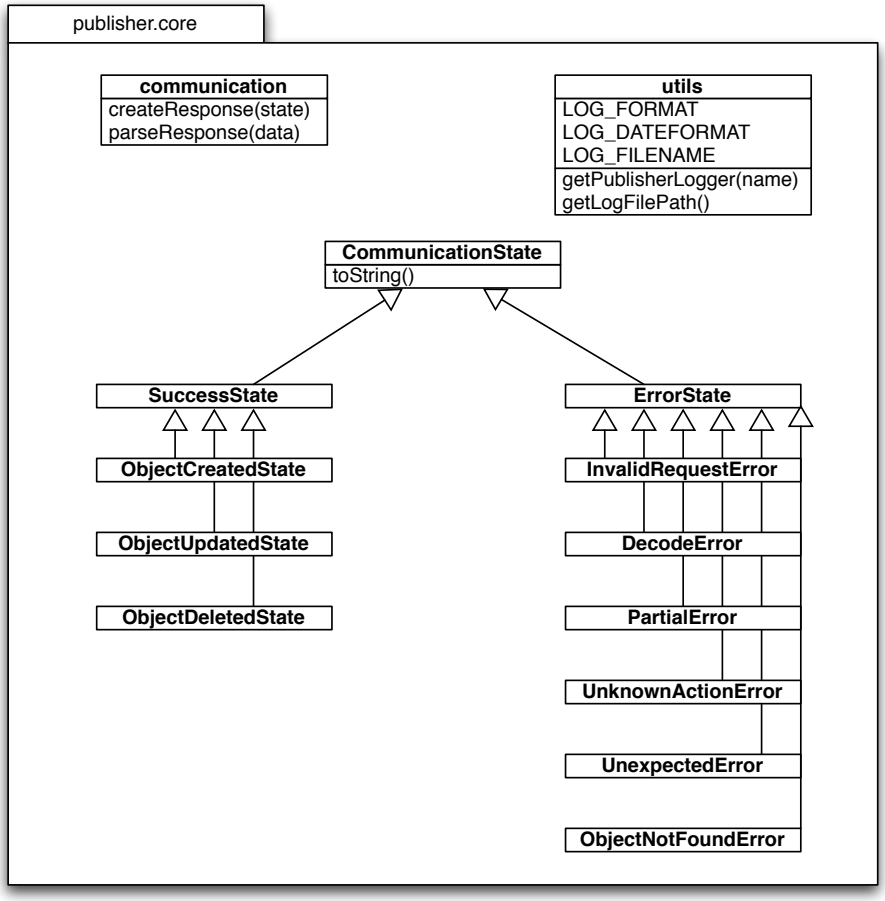

Abbildung 10: Klassendiagramm publisher.core

15.1.2. Paket publisher.sender

Dieses Paket wird nur auf der Quell-Instanz installiert. Es ist das grösste Paket und stellt folgende Funktionalität zur Verfügung:

- **Queue**: Die Warteschleife (Queue) befindet sich auf der Quellinstanz und dient dazu, dass die Anfragen asynchron abgearbeitet werden können. Sie besteht aus einem Zope3-Adapter Objekt, das eine Liste von Jobs führt.
- **Job**: Soll ein Plone-Objekt publiziert oder gelöscht werden, wird ein Job-Objekt erstellt. Dieses Job-Objekt wird anschliessend in der Queue abgelegt. Wenn die Queue abgearbeitet wird, werden die Daten mit der entsprechenden Aktion über eine HTTP-Anfrage an das Zielsystem gesendet.

- **Konfiguration**: Die Konfiguration ist ein Objekt, das - ähnlich wie die Queue - als Adapter registriert ist. Die Konfiguration enthält primär eine Liste von Ziel-Instanzen. Pro Ziel-Instanz werden Informationen wie Adresse und Zugangsdaten gespeichert.
- **Extraktor**: Das Extraktor-Modul ist dafür zuständig, alle wichtigen Informationen aus einem Plone-Objekt zu extrahieren und Python-Dictionary zu erstellen. Diese Daten können anschliessend von der JSON-Bibliothek so in ein JSON-String konvertiert werden.
- **Aktionen / Views**: Das Paket enthält Aktionen (sog. Views) zum Erstellen eines Jobs und zum Abarbeiten der Queue.
- **Konfigurationsseiten**: Das Paket enthält ein sog. „Configlet“. Das ist eine Sammlung von Konfigurationsseiten für das Produkt. Diese Konfigurationsseiten sind als Configlet in der Plone-Instanz registriert. Auf den Konfigurationsseiten kann man die Ziel-Instanzen konfigurieren und die Warteschleife steuern (leeren, ausführen, betracheten).

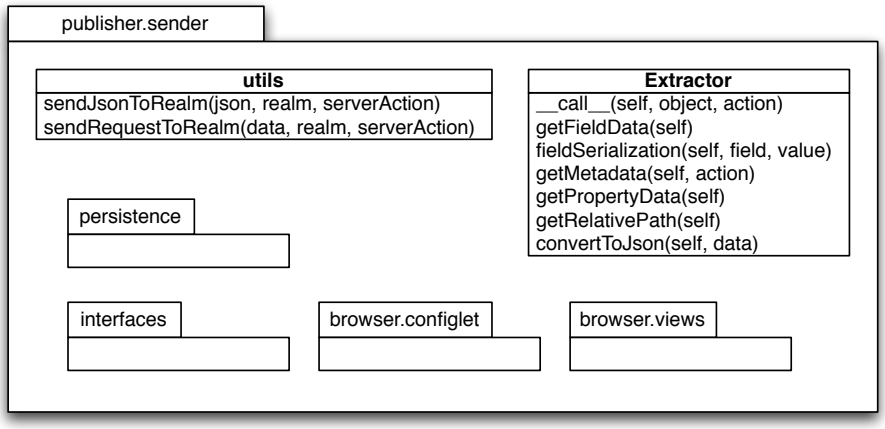

Abbildung 11: Klassendiagramm publisher.sender

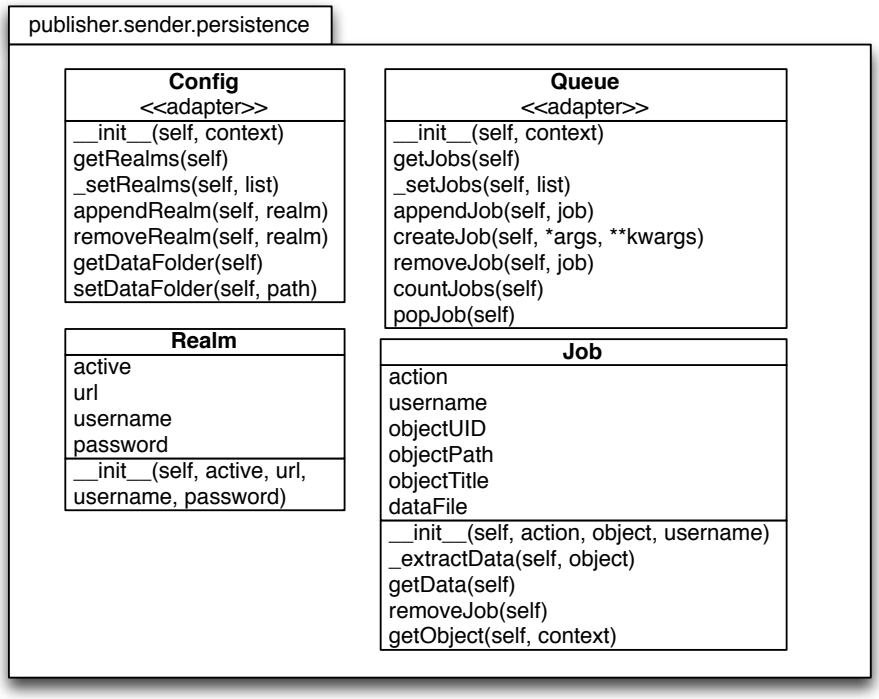

Abbildung 12: Klassendiagramm publisher.sender.persistence

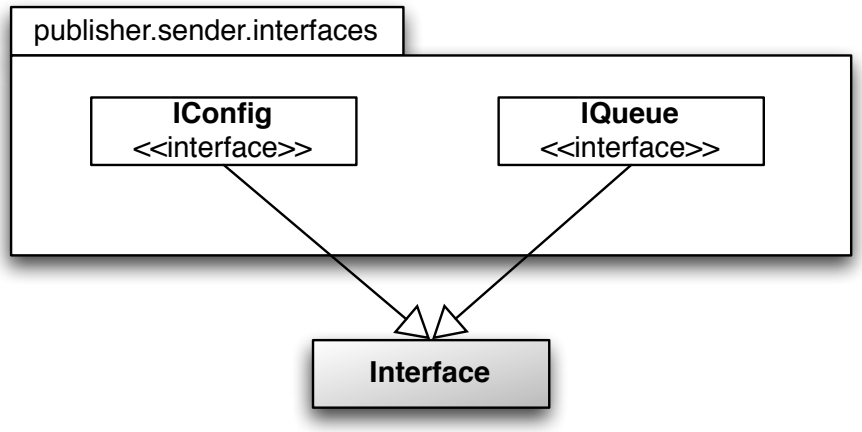

Abbildung 13: Klassendiagramm publisher.sender.interfaces

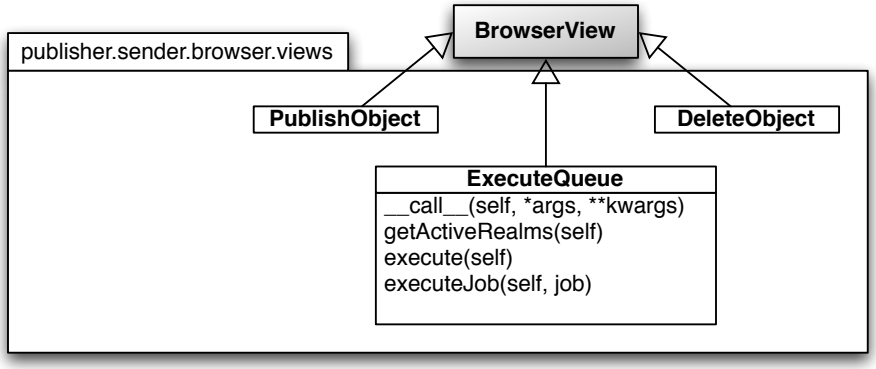

Abbildung 14: Klassendiagramm publisher.sender.browser.views

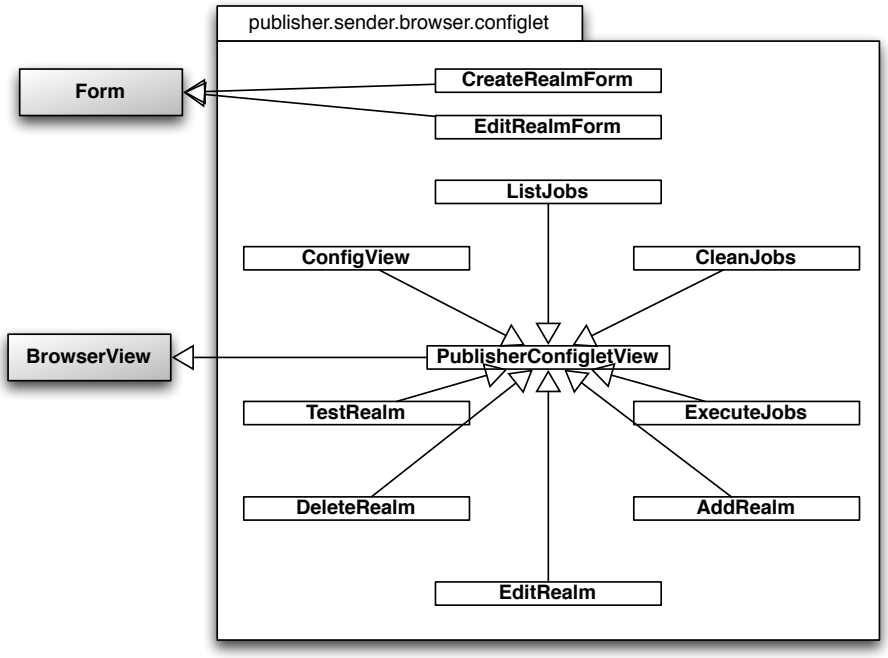

Abbildung 15: Klassendiagramm publisher.sender.browser.configlet

15.1.3. Paket publisher.receiver

Das Paket publisher.receiver wird auf der Empfänger-Instanz installiert. Es ist sehr klein gehalten und enthält folgende Komponenten:

- **TestConnectionView**: Diese View wird vom Configlet (Paket *publisher.sender*) benutzt, um die Verbindung zu testen.
- **ReceiveObjectView**: Diese View wird vom Sender pro Job aufgerufen. Als Parameter (jsondata) werden die mit JSON formatierten Daten des Jobs übergeben. Diese werden von dieser View grundsätzlich verarbeitet.

- **Decoder**: Wenn eine Job-Anfrage vom Sender eintrifft, leitet die *ReceiveObject*-View die JSON-Daten an eine Decoder-Instanz weiter. Diese entpackt die JSON-Daten und validiert sie auf Vollständigkeit. Dabei werden die Daten auch mit dem Archetypes-Schema des betreffenden Inhaltstyps abgeglichen und allenfalls Anpassungen gemacht. Schlussendlich werden die entpackten Daten zurück an die *ReceiveObject*-View gesendet, welche anschliessend die erwünschten Aktionen durchführt und Objekte erstellt, aktualisiert oder löscht.

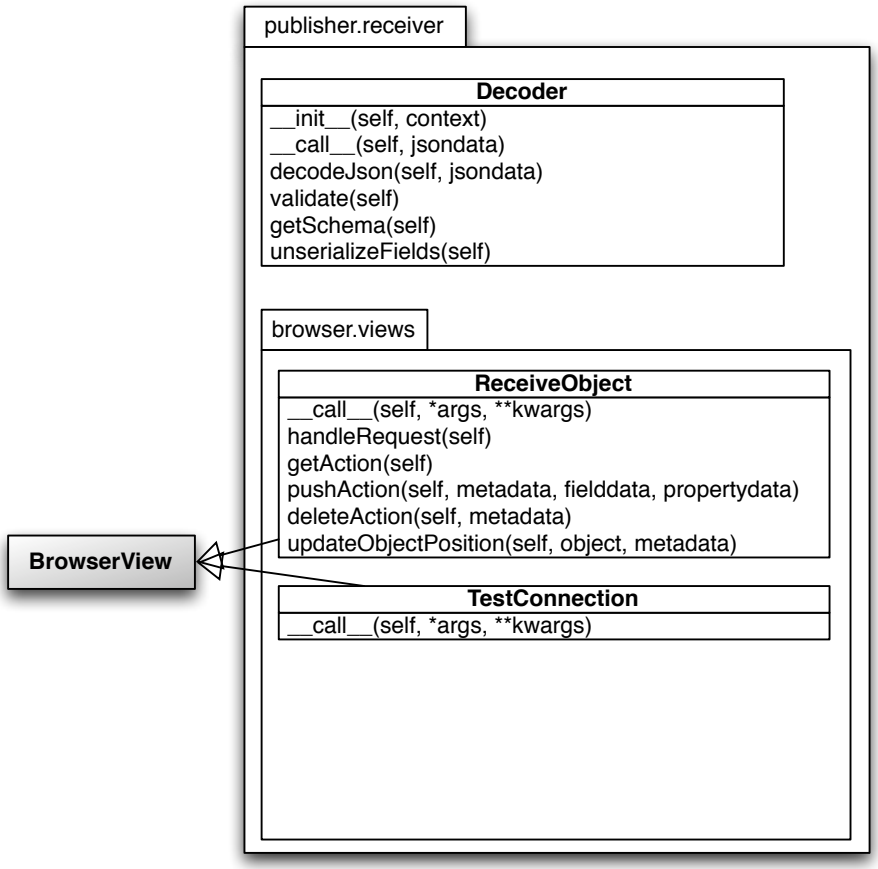

Abbildung 16: Klassendiagramm publisher.receiver

15.2. Probleme während der Realisierung

Während der Realisierung hatte ich diverse Probleme, die ich zu lösen hatte. Hier erkläre ich die grössten und wichtigsten Probleme und wie ich sie gelöst habe.

15.2.1. Setzen der Unique-ID

Plone vergibt allen Objekten eine Unique-ID (UID), mit welcher es die Objekte erkennen und eindeutig adressieren kann. Zum Beispiel die Referenzen (ReferenceField) basieren direkt auf den UIDs.

Diese UIDs werden automatisch generiert und können beim Erstellen eines Objektes nicht übergeben werden. Nach einer Recherche fand ich die Methode _setUID() welche die UID eines Objektes neu setzt und auch gleich die verschiedenen Kataloge anpasst.

Nachfolgend eine gekürzte Version des Vorgangs zum Erstellen eines neuen Objektes auf der Zielinstanz:

Listing 3: Erstellen eines neuen Objektes (gekürzt und angepasst)

```
"""
 container     : der Ordner, in dem das Objekt erstellt werden
                 soll
 portalType    : Name des zu erstellenden Typs (z.B. Image)
 id            : ID des zu erstellenden Objektes
 uid           : UID (gem. Quellobjekt)
"""
container.invokeFactory(portalType, id)
object = container.get(id)
object._setUID(uid)
object.processForm()
```

15.2.2. Übertragung binärer Daten

Binäre Daten wie Bilder und Dateien werden von Plone direkt in der ZODB, der Zope-Datenbank, gespeichert. Sie werden mit den entsprechenden Attributen bzw. den dazugehörenden Methoden (Accessor, Mutator) gesetzt und wieder ausgelesen.

Da diese Datei- und Bild-Felder ein wichtiger Bestandteil der Plone-Objekte sind und in vielen Standard-Inhaltstypen verwendet werden, muss das publisher-Produkt unbedingt damit umgehen und binäre Daten korrekt publizieren können.

Der kritische Punkt mit binären Daten ist die Übertragung. Das Format JSON ist laut Spezifikation [11] nicht fähig, binäre Daten zu übetragen. Neben den verschachtelten Typen (Listen, Dictionaries bzw. sog. Objekte) sind nur primitive Datentypen wie Zeichenkette, Zahlen, Boolean oder "Null"gültig.

Also müssen die binären Daten entweder auf eine andere Art und Weise übertragen werden, oder sie müssen in einen gültigen Typ umgewandelt werden. Da eine weitere Übertragungsart umständlich und fehleranfällig ist und das ganze Produkt an unnötiger Komplexität zunehmen würde, habe ich beschlossen, die binären Daten mit dem Algorithmus **base64** zu

[11] http://www.json.org/

codieren. So erhalte ich eine Zeichenkette (string), was ein gültiger Typ ist und trotzdem wieder in die originalen binären Daten zurückgewandelt werden kann.

Nachteil Der grosse Nachteil dabei ist dass das mit einer base64-Verschlüsselung das Datenvolumen um etwa ein Drittel zunimmt. Bei grossen Dateien kann das grosse Auswirkungen auf die Dauer der Übertragung haben. Da die Übertragung aber asynchron geschieht und der Benutzer also nicht warten muss, ist dieses Problem vernachlässigbar.

Listing 4: Codierung der binären Daten mit base64 (extractor.py)

```
    def fieldSerialization(self, field, value):
        """
        Custom serialization for fields which provide field values that are incompatible
        with simplejson / JSON-standard.
        field : Field-Object from Schema
        value : Return-Value of the Raw-Accessor of the Field on the current context
        Return : JSON-optimized value
        """
        # DateField : returns a DateTime-Object as value. We cast it to string and it
        #   looks like '2010-05-31 13:06:01.925652'
        if isinstance(field, DateTimeField) and value:
            value = str(value)
        # FileField : returns a File-Object. The binary data can be accessed with the
        #   attribute "data". We encode it with base64 to produce a string
        elif isinstance(field, FileField):
            if not isinstance(value, str):
                value = value.data
            value = base64.encodestring(value)
        return value
```

15.2.3. Schema eines Typs holen

In Plone erhalten alle Inhaltstypen ein Schema, welches die Felder enthält, die in Objekten dieses Typs zur verfügung stehen. Das publisher-Produkt basiert ganz auf diesem Schema. Das Schema steht in jedem Objekt unter dem Attribute ßchemaßur Verfügung.

Bei einer Anfrage an die Zielinstanz muss diese als erstes die Anfrage entschlüsseln und überprüfen, ob diese gültig ist. Dazu gehört das validieren der Felder und allenfalls das decodieren der Werte.

Wenn jetzt bei einem Request ein Objekt publiziert wird, das bis anhin auf der Zielinstanz noch nicht existiert (und auch kein anderes Objekt vom gleichen Typ), wie kann da das Schema geholt werden?

Mit diesem Problem habe ich ein Weile gekämpft und bin dann mit der Unterstützung von Mathias Leimgruber (Plone-Entwickler 4teamwork) zu folgender Lösung gekommen:

Listing 5: Archetypes-Schema eines Typs auslesen

```
typename = 'ATFolder'
typeinfo = context.portal_types.getTypeInfo(typename)
klass = filter(lambda x:x['portal_type']==typename, context.archetype_tool.
    listRegisteredTypes())[0]['klass']
schema = klass.schema
```

15.2.4. Referenzen in Rich-Text

Plone benutzt als WYSIWYG-Editor das freie Software Modul Kupu. Mit Kupu kann aus einem Text ein Link auf ein anderes Plone-Objekt erstellt werden. Dabei speichert Kupu (je nach konfiguration) die UID des Objektes. Ein so erstellter Link könnte wie folgt aussehen:

Listing 6: Beispiellink Intern

```
1 <a href="resolveuid/d4f89fbd82ffeecbe0d11032f63be19b">Impressum</a>
```

Beim Aufrufen der Seite wird der Link dann in eine „richtige“ Adresse umgewandelt:

Listing 7: Beispiellink Ausgabe

```
1 <a href="/seite/impressum">Impressum</a>
```

Wenn man mit dem normalen *Accessor* des Feldes auf den Text zugreift, so erhält man bereits den konvertierten Text. Dies ist für das publisher-Produkt problematisch, denn diese Adresse muss nicht unbedingt stimmen: wenn die Adresse (ohne Domain) der Quellinstanz nicht mit der Adresse der Zielinstanz übereinstimmt, so wird dieser Link nicht funktionieren.

Lösung Wird die Methode *getRaw* anstatt des normalen Accessors verwendet, so erhält man den Wert, wie er effektiv in der Datenbank gespeichert ist.

Listing 8: Aufruf und Ausgabe über Accessor

```
1 >>> accessor = context.getField('text').getAccessor(context)
2 >>> accessor()
3 '<a href="/seite/impressum">Impressum</a>'
```

Listing 9: Aufruf und Ausgabe über getRaw

```
1 >>> context.getField('text').getRaw(context)
2 '<a href="resolveuid/d4f89fbd82ffeecbe0d11032f63be19b">Impressum</a>'
```

15.2.5. Bilder / Dateien löschen

Durch das Kodieren binärer Daten wie Bilder oder Dateien können diese korrekt übertragen werden. Doch das Löschen der Dateien wurde zur Herausforderung: Wie auch bei anderen Feldern kann der Wert bei einem File- oder ImageField mit dem entsprechenden Mutator (Setter-Methode) auf dem Objekt gesetzt werden. Ist aber der Wert des Parameters beim Mutator-Aufruf negativ (z.B. leerer String, None oder False), so wird nichts geändert. Will man aber das Bild oder die Datei löschen, so muss man bei einem FileField „DELETE_FILE“ beim Mutator-Aufruf übergeben, bei einem ImageField aber „DELETE_IMAGE“.

Um dieses Problem zu beseitigen, musste ich beim Empfänger eine entsprechende Abfrage einbauen:

Listing 10: Bilder/Dateien löschen (decoder.py)

```
class Decoder(object):
    """
    Decodes json data to dictionary and validates it.
    It also validates and decodes all schema field values.
    """

    # ...

    def unserializeFields(self):
        schema = self.getSchema()
        for field in schema.fields():
            name = field.getName()
            # ...
            # ImageField: treat empty files special
            if isinstance(field, ImageField):
                if len(self.data['fielddata'][name])==0:
                    self.data['fielddata'][name] = 'DELETE_IMAGE'
            # FileField (direct): treat empty files special
            if field.__class__==FileField:
                if len(self.data['fielddata'][name])==0:
                    self.data['fielddata'][name] = 'DELETE_FILE'
```

16. Test

16.1. Testverfahren

Die Tests werden manuell durchgeführt. Es wird eine Reihe von Testfällen definiert, die am Ende der IPA getestet werden. Grundsätzlich ist es nicht vorgesehen, die fehlgeschlagenen Tests direkt zu korrigieren. Sie werden ausreichend dokumentiert und nach der IPA korrigiert. Kleinere Probleme, die schnell gelöst werden können, können auch während der IPA korrigiert werden.

Nach jedem Test-Case muss vor dem Überprüfen des Resultats die Queue ausgeführt werden!

16.2. Bedingungen

Bevor die Tests durchgeführt werden, muss das System wie folgt konfiguriert sein:

- Ziel-Instanz
  - Plone-Instanz (>=3.1.1,<3.2dev) installiert
  - publisher.core installiert
  - publisher.receiver installiert
- Quell-Instanz
  - Plone-Instanz (>=3.1.1,<3.2dev) installiert
  - publisher.core installiert
  - publisher.sender installiert
  - publisher.demo installiert
  - Ziel-Instanz als Empfänger konfiguriert

16.3. Testfälle / Testprotokoll

Nr	Aktion	Erwartung	Ergebnis
1	Auf Quell-Instanz ein „Ordner“-Objekt erstellen (Titel: *Ordner1*)	Das Objekt erscheint nicht auf der Ziel-Instanz	OK
2	*Quell-Instanz:* Ordner1 publizieren (Workflow)	*Ziel-Instanz:* Das Objekt erscheint und ist öffentlich sichtbar. *Quell-Instanz:* Objekt nicht mehr bearbeitbar	OK
3	*Quell-Instanz:* Ordner1 revidieren (Workflow)	*Ziel-Instanz:* sichtbar *Quell-Instanz:* bearbeitbar	OK

Nr	Aktion	Erwartung	Ergebnis
4	*Quell-Instanz:* Ordner1 Titel ändern zu *Ordner1-anders*	*Ziel-Instanz:* Titel immer noch *Ordner1*	OK
5	*Quell-Instanz:* Ordner1 publizieren (Workflow)	*Ziel-Instanz:* Titel nun *Ordner1-anders*	OK
6	*Quell-Instanz:* Ordner1 zurückziehen (Workflow)	*Ziel-Instanz:* Objekt existiert nicht mehr	OK
7	*Quell-Instanz:* Ordner1 erneut publizieren (für weitere Tests)	*Ziel-Instanz:* sichtbar	OK
8	*Quell-Instanz:* Neues „Seite“-Objekt (Titel: *Seite* erstellen und mit Inhalt füllen. Anschliessend Objekt publizieren	*Ziel-Instanz:* Objekt erscheint und enthält alle Daten	OK
9	*Quell-Instanz:* In *Ordner1-anders* ein „Bild“-Objekt erstellen und ein Bild hochladen. Anschliessend Bild veröffentlichen.	*Ziel-Instanz:* Bild-Objekt erscheint und enthält das hoch geladene Bild	OK
10	*Quell-Instanz:* Objekt *Seite* revidieren (Workflow) Im TextFeld (Kupu) das Bild einfügen. Das *Seite*-Objekt publizieren	*Ziel-Instanz:* Das *Seite*-Objekt zeigt das Bild an	OK
11	*Quell-Instanz:* Bild revidieren Bild bearbeiten: Option „Von Navigation ausschließen“ aktivieren Bild publizieren	*Ziel-Instanz:* Bild erscheint nicht mehr in Navigation, existiert aber noch	OK
12	*Quell-Instanz:* *Ordner1-anders* revidieren Darstellung -> „Artikel aus dem Ordner ...“ -> *Seite* auswählen Ordner publizieren	*Ziel-Instanz:* Bei Klick auf den Ordner in der Navigation erscheint der Inhalt der *Seite*	OK
13	*Quell-Instanz: Seite*: Verweis auf *Ordner1-anders* hinzufügen Seite publizieren	*Ziel-Instanz:* Auf der *Seite*-Ansicht erscheint der Verweus auf *Ordner1-anders*	OK

Nr	Aktion	Erwartung	Ergebnis
14	*Quell-Instanz:* Vier Objekte vom Typ „Seite“ anlegen. (Titel: Objekt A, Objekt B, Objekt C, Objekt D) Erstellte Objekte publizieren	*Ziel-Instanz:* Beim Aufruf des Ordners erscheinen die Objekte in korrekter Reihenfolge (A, B, C, D)	OK
15	*Quell-Instanz:* Auf Ordner, in welchem die vier Objekte sind, auf „Inhalte“ gehen und Sortierreihenfolge ändern zu: A,C,B,D Die vier Objekte publizieren	*Ziel-Instanz:* Reihenfolge der Objekte in Navigation wurde angepasst und ist jetzt: A,C,B,D	OK
16	*Quell-Instanz:* Objekte A,B,C,D zurückziehen Objekte A und D publizieren Objekt C publizieren	*Ziel-Instanz:* Reihenfolge: A,C,D	OK
17	*Quell-Instanz:* Neues „Bild“-Objekt erstellen und Bild hochladen Bild-Objekt publizieren	*Ziel-Instanz:* Bild-Objekt erscheint und enthält Bild	OK
18	*Quell-Instanz:* Bild-Objekt revidieren Bild-Objekt bearbeiten und in Bild-Feld das „Lösche aktuelles Bild“ wählen Bild-Objekt publizieren	*Ziel-Instanz:* Bild-Objekt enthält kein Bild mehr	Fehlgeschlagen

Tabelle 16: Testfälle / Testprotokoll

Der Test Nr. 18 ist Fehlgeschlagen, weil in einem Bild-Objekt das Bild-Feld auf obligatorisch gesetzt ist. Die Option „Lösche aktuelles Bild“ ist also deaktiviert. Die Tests Nr. 17 und Nr. 18 wurden mit dem Inhaltstyp „Nachricht“ wiederholt. Eine Nachricht kann auch ein Bild enthalten, das Bild ist aber nicht obligatorisch. Das Resultat war wie gewünscht: Das Bild konnte wieder entfernt werden.

16.4. Unterschrift Testprotokoll

Datum	Name	Unterschrift
6.03.2009	Jonas Baumann	

Tabelle 17: Unterschriften Testprotokoll

Teil III.

Anhang

A. Glossar

A

Archetypes Archetypes vereinfacht das Erstellen von neuen Inahaltstypen in Plone. Um einen Inhaltstyp zu erstellen muss man mit Archetypes lediglich ein Schema und eine Klasse definieren, die benötigten Standard-Ansichten (Erstellen, Bearbeiten, Anzeigen, Löschen, Berechtigungen Verwalten, etc) werden von Archetypes generisch zur Verfügung gestellt. (http://plone.org/products/archetypes), S. 12.

B

Buildout Werkzeug für das automatisierte Installieren von Python Paketen und Applikationen. (http://pypi.python.org/pypi/zc.buildout), S. 16.

C

CMS Content Management System (Inhaltsverwaltungssystem): Ein System zum Erstellen und verwalten von Inhalten, meist für Webbasierte Anwendungen, S. 8.

Content Staging Unter Content Staging versteht man in der Informatik einen Prozess der Informationsintegration, in dem Daten in einem Datenbereich (engl.: staging area, der Bereitstellungsraum oder Sammelplatz beim Militär) temporär zwischengespeichert werden, um sie dort zu bereinigen und zu transformieren. (Quelle: http://de.wikipedia.org/), S. 8.

I

IPA Individuelle Praktische Arbeit, S. 5.

J

JSON (JavaScript Object Notation) ist ein Format zur Übertragung von Informationen in einer von Mensch und Maschine lesbaren Form. Spezifikation unter http://www.json.org, S. 47.

K

Kupu Kupu ist der WYSIWYG-Editor, der in Plone standardmässig enthalten ist., S. 49.

P

Paster Paster ist ein Werkzeug um das Grundgerüst für Plone3-Buildouts und Plone3-Paketen zu erstellen., S. 16.

Plone Content Management System (siehe http://www.plone.org), S. 8.

Python Python ist eine dynamische, objektorientierte Programmiersprache (siehe http://www.python.org), S. 8.

W

WYSIWYG-Editor (What You See Is What You Get) Web: Visueller Editor zum Erfassen und Gestalten von Inhalten. (siehe auch http://de.wikipedia.org/wiki/WYSIWYG), S. 49.

Z

ZODB Zope Object Database. In der ZODB werden dynamisch veränderbare Inhalte abgelegt. Die ZODB enthält alle vorhandenen Objekte in einem hierarchischen Baum und bildet die Grundlage der Akquisition. (URL: http://wiki.zope.org/ZODB/FrontPage), S. 8.

Zope (*Z Object Publishing Environment*) ist ein freier Anwendungsserver, geschrieben in Python. (http://www.zope.org/), S. 8.

B. Abbildungsverzeichnis

C. Tabellenverzeichnis

D. Quellenverzeichnis

Bücher

- *LaTeX- Eine Einführung* (Helmut Kopka)
  1988, Addison-Wesley (Deutschland) GmbH
  ISBN: 3-89319-199-2
- *Der LaTeX Begleiter* (Frank Mittelbach, Michel Goossens)
  2. Auflage 2005, Pearson Studium
  ISBN: 978-3-8273-7166-9
- *KOMA-Script* (Markus Kohm, Jens-Uwe Morawski)
  2. Auflage 2006
  ISBN: 3-86541-089-8
- *Professional Plone Development* (Martin Aspeli)
  2007, Packt Publishing
  ISBN: 978-1-847191-98-4
- *Qualifikationsverfahren Informatik - Kanton Bern 2009* (Andy Bula, Benedikt Sutter, Walter Aeby)
  2009, Verbandsprüfungskomission VPK Bern

Webseiten

- `http://www.python.org`
  Offizielle Webseite der dynamischen, objektorientierten Sprache Python
- `http://www.plone.org`
  Offizielle Webseite des Open Source Content Management Systems Plone
- `http://www.zope.org`
  Offizielle Website des Applikationsservers Zope
- `http://www.json.org`
  Offizielle Webseite des Übertragungsstandards JSON
- `http://www.python.org/dev/peps/pep-0008/`
  Offizielles Python Style Guide
- `http://epydoc.sourceforge.net/epydoc.html`
  Epydoc Standard für API-Dokumentation in Python-Code

Weitere Adressen sind in der Dokumentation als Fussnoten vorhanden.

E. Quelltext

In diesem Abschnitt folgen Auszüge aus dem Quelltext. Es werden nicht alle Dateien abgebildet, da nicht der ganze Quelltext für das Verständnis nötig sind. Der Dateikopf ist nur in der ersten Datei enthalten und wurde bei allen weiteren Dateien entfernt, um Platz zu sparen.

Dateistruktur

Um ein besseres Verständnis zu ermöglichen, folgt hier die Auflistung aller Dateien, die erstellt wurden (inkl. automatisch generierte Dateien).

Modul publisher.core

```
setup.py
publisher/
    __init__.py
    core/
        __init__.py
        communication.py
        configure.zcml
        states.py
        utils.py
```

Modul publisher.sender

```
setup.py
publisher/
    __init__.py
    sender/
        __init__.py
        configure.zcml
        dependencies.zcml
        extractor.py
        interfaces.py
        persistence.py
        profiles.zcml
        utils.py
        browser/
            __init__.py
            addRealm.pt
            configlet.py
            configure.zcml
            editRealm.pt
            executeJobs.pt
            interfaces.py
            listJobs.pt
            publisherConfig.pt
            views.py
        profiles/
            default/
                controlpanel.xml
                metadata.xml
```

Modul publisher.receiver

```
setup.py
publisher/
    __init__.py
    receiver/
        __init__.py
        configure.zcml
        decoder.py
        browser/
            __init__.py
            configure.zcml
            views.py
```

Modul publisher.core

Listing 11: states.py

```
#
# File:     states.py
# Author:   Jonas Baumann <j.baumann@4teamwork.ch>
# Modified: 06.03.2009
#
# Copyright (c) 2009 by 4teamwork.ch
#
# GNU General Public License (GPL)
#
# This program is free software; you can redistribute it and/or
# modify it under the terms of the GNU General Public License
# as published by the Free Software Foundation; either version 2
# of the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
# 02110-1301, USA.
#
__author__ = """Jonas Baumann <j.baumann@4teamwork.ch>"""


# Superclass

class CommunicationState(Exception):
    """
    Superclass for all states used by publisher packages.
    The CommunicationState class inherits from Exception, because
    we want to be able to "raise" a CommunicationState-object.

        >>> raise CommunicationState('test')
        Traceback (most recent call last):
          File "<stdin>", line 1, in <module>
        states.CommunicationState: test
    """

    def __init__(self, message='', *args, **kwargs):
        """
        Constructs the CommunicationState object. If a message
        is provided, it will be stored for later use.

        @param message:     Status message (optional)
        @type message:      string
```

```
49          """
50          Exception.__init__(self, message, *args, **kwargs)
51          self.message = str(message)
52
53      def toString(self):
54          """
55          Converts a CommunicationState object to a string for printing
56          or for sending to a other instance.
57
58              >>> print CommunicationState('Hello World').toString()
59              CommunicationState
60              Hello World
61
62
63          @rtype:             string
64          @return:            string containing classname and message
65          """
66          return '\n'.join([
67                  self.__class__.__name__,
68                  self.message,
69          ])
70
71  # Succesful states
72
73  class SuccessState(CommunicationState):
74      """
75      Superclass for all states which indicate that the communication was
76      successful.
77      """
78
79
80  class ObjectCreatedState(SuccessState):
81      """
82      Indicates that a new object was created.
83      """
84
85
86  class ObjectUpdatedState(SuccessState):
87      """
88      Indicates that the object was updated successfully.
89      """
90
91  class ObjectDeletedState(SuccessState):
92      """
93      Indicates that the object was removed successfully.
94      """
95
96
97  # Failed states
98
99  class ErrorState(CommunicationState):
100     """
101     Superclass for all states which indicate that the communication was
102     not successful.
103     """
104
105
106 class InvalidRequestError(ErrorState):
107     """
108     Indicates a problem with the submitted request.
109     """
110
111
112 class DecodeError(ErrorState):
113     """
114     Indicates that the decoder was not able to decode the request data to
115     json.
116     """
117
118 class PartialError(ErrorState):
```

```
119         """
120         Indicates that the decoded data was not complete, there were parts missing.
121         """
122
123     class UnknownActionError(ErrorState):
124         """
125         An UnknownActionError is raised if the action is not one of the defined
126         actions (push, delete).
127         """
128
129     class UnexpectedError(ErrorState):
130         """
131         Any exception which is not catched will raise a UnexpectedError containing
132         the Exception information.
133         """
134
135     class ObjectNotFoundError(ErrorState):
136         """
137         Indicates that the object could not be found.
138         """
```

Listing 12: utils.py

```
1   # ... header wurde entfernt ...
2
3   __author__ = """Jonas Baumann <j.baumann@4teamwork.ch>"""
4
5   import os.path
6   import logging
7
8   from ZConfig.components.logger import loghandler
9
10  """
11  @var LOG_FORMAT:        Logging Format (see python logging module)
12  """
13  LOG_FORMAT = '%(asctime)s %(levelname)s [%(name)s] %(message)s'
14
15  """
16  @var LOG_DATEFORMAT:    Logging Dateformat (see python logging module)
17  """
18  LOG_DATEFORMAT = '%Y-%m-%dT%H:%M:%S'
19
20  """
21  @var LOG_FILENAME:      Filename for the log-file to log to.
22  """
23  LOG_FILENAME = 'publisher.log'
24
25  """
26  @var logHandler:        FileHandler instance (see python logging module)
27  """
28  logHandler = None
29
30  def getPublisherLogger(name):
31      """
32      Creates a python logger which loggs into the custom publisher log.
33
34      @param name:    Name of the logger instance (see python logging module)
35      @type name:     string
36      @return:        Python logging object
37      """
38      # get logger
39      logger = logging.getLogger(name)
40      global logHandler
41      if not logHandler:
42          # create new handler
43          filepath = getLogFilePath()
44          if filepath:
45              logHandler = logging.FileHandler(filepath)
46              # register formatter
```

```
47              logHandler.setFormatter(logging.Formatter(fmt=LOG_FORMAT,
48                      datefmt=LOG_DATEFORMAT))
49      if logHandler and logHandler not in logger.handlers:
50          # register it
51          logger.addHandler(logHandler)
52      return logger
53
54  def getLogFilePath():
55      """
56      Returns the log file path for the publisher log file.
57      Returns None if it cannot guess the default log file path.
58
59      @return:        logfile path or None
60      """
61      # get default log file path
62      for handler in logging.root.handlers:
63          if isinstance(handler, loghandler.FileHandler):
64              path = os.path.dirname(handler.baseFilename)
65              return os.path.join(path, LOG_FILENAME)
66      return None
```

Modul publisher.receiver

Listing 13: views.py

```
1   # ... header wurde entfernt ...
2
3   __author__ = """Jonas Baumann <j.baumann@4teamwork.ch>"""
4
5   # python imports
6   import traceback, sys, os.path
7
8   # zope imports
9   from Products.Five import BrowserView
10
11  # plone imports
12  from Products.CMFPlone.interfaces import IPloneSiteRoot
13
14  # publisher imports
15  from publisher.receiver import decoder
16  from publisher.receiver import getLogger
17  from publisher.core import states, communication
18
19  class ReceiveObject(BrowserView):
20      """
21      The ReceiveObject View is called be publisher.sender module. It expects a
22      "jsondata"-paremeter in the request. It parses the jsondata and runs the
23      specified action.
24      """
25
26      def __call__(self, *args, **kwargs):
27          """
28          @return:    response string containing a representation  of a
29                      CommunicationState as string.
30          """
31          # get a logger instance
32          self.logger = getLogger()
33          try:
34              state = self.handleRequest()
35          except Exception, e:
36              if isinstance(e, states.CommunicationState):
37                  # if a CommunicationState is raised, we will use it as response
38                  state = e
39              else:
40                  # otherwise we encapsulate the exception in a UnexpectedError
41                  exc = ''.join(traceback.format_exception(*sys.exc_info()))
42                  state = states.UnexpectedError(exc)
```

```
43              self.logger.error('request failed: %s' % (state.toString()))
44          # get the string representation of the CommunicationState
45          resp = communication.createResponse(state)
46          try:
47              # if a exception was thrown, we add the traceback to the response
48              resp += ''.join(traceback.format_exception(*sys.exc_info()))
49          except:
50              pass
51          return resp
52
53      def handleRequest(self):
54          """
55          Handles a request from publisher.sender. It parses the paremeter
56          "jsondata" which should be in the request.
57
58          @raise:     CommunicationState
59          @return:    CommunicationState-Object
60          """
61          # get the jsondata
62          jsondata = self.request.get('jsondata', None)
63          self.logger.info('Receiving request (data length: %i Byte)' % len(jsondata))
64          if not jsondata:
65              raise states.InvalidRequestError('No jsondata provided')
66          # decode the jsondata to a python dictionary
67          self.data = decoder.Decoder(self.context)(jsondata)
68          # get the action type ..
69          action = self.getAction()
70          # .. and run the action specific method ..
71          if action=='push':
72              metadata = self.data['metadata']
73              fielddata = self.data['fielddata']
74              propertydata = self.data['properties']
75              return self.pushAction(metadata, fielddata, propertydata)
76          elif action=='delete':
77              metadata = self.data['metadata']
78              return self.deleteAction(metadata)
79          else:
80              # ... or raise a UnexpectedError
81              raise states.UnknownActionError()
82          # should not get here
83          raise states.UnexpectedError()
84
85      def getAction(self):
86          """
87          Returns the action name of the current request.
88
89          @rtype:     string
90          @return:    action name ('push' or 'delete')
91          """
92          return self.data['metadata']['action']
93
94      def pushAction(self, metadata, fielddata, propertydata):
95          """
96          Is called if the action is "push". It creates or updates a object
97          (identifiers in metadata) with the given fielddata.
98          For more infos about the provided data see publisher.sender.extrator
99
100         @param metadata:        metadata dictionary containing UID, portal_type,
101                                 action, physicalPath and sibling_positions
102         @type metadata:         dict
103         @param fielddata:       fielddata dictionary contains the values of the
104                                 archetype-fields of this object
105         @type fielddata:        dict
106         @param propertydata:    propertydata is a list of property-dictionaries
107         @type propertydata:     list
108         @return:                CommunicationState instance
109         @rtype:                 publisher.core.states.CommunicationState
110         """
111         # do we have to update or create? does the object already exist?
112         # ... try it with the uid
```

```
113             absPath = self._getAbsolutePath(metadata['physicalPath'])
114             object = self._getObjectByUID(metadata['UID'])
115             if object:
116                 # ... is it the right object?
117                 if '/'.join(object.getPhysicalPath())!=absPath:
118                     raise states.UnexpectedError('UID already used or object in ' +\
119                             'wrong place')
120             # create the object if its not existing ...
121             new_object = False
122             if not object:
123                 self.logger.info(
124                         'Object with UID %s does not existing: creating new object'%(
125                                 metadata['UID'],
126                         )
127                 )
128                 # ... find container
129                 container = self._findContainerObjectByPath(absPath)
130                 if not container:
131                     raise states.ErrorState('Could not find container of %s' %
132                             absPath)
133                 self.logger.info('... container: "%s" at %s' % (
134                         container.Title(),
135                         '/'.join(container.getPhysicalPath()),
136                 ))
137                 # ... create object
138                 object = container.get(container.invokeFactory(
139                         metadata['portal_type'],
140                         fielddata['id'],
141                 ))
142                 object._setUID(metadata['UID'])
143                 object.processForm()
144                 new_object = True
145             # update with new values
146             self.logger.info('Updating object values (UID %s)' %
147                     metadata['UID'])
148             for field in object.schema.fields():
149                 fieldname = field.getName()
150                 # do not update "id" field
151                 if fieldname in fielddata.keys() and fieldname not in ['id']:
152                     field.getMutator(object)(fielddata[fieldname])
153             # update properties
154             self._updateProperties(object, propertydata)
155             # set object position
156             self.updateObjectPosition(object, metadata)
157             # finalize and reindex
158             object.processForm()
159             object.reindexObject()
160             # return the appropriate CommunicationState
161             if new_object:
162                 return states.ObjectCreatedState()
163             else:
164                 return states.ObjectUpdatedState()
165 
166         def deleteAction(self, metadata):
167             """
168             Deletes the object identified by metadata values.
169 
170             @param metadata:        metadata dictionary containing UID, portal_type,
171                                     action, physicalPath and sibling_positions
172             @type metadata:         dict
173             @return:                CommunicationState instance
174             @rtype:                 publisher.core.states.CommunicationState
175             """
176             # find the object
177             object = self._getObjectByUID(metadata['UID'])
178             if not object:
179                 raise states.ObjectNotFoundError()
180             # delete the object
181             self.logger.info('Removing object with UID %s' % metadata['UID'])
182             container = object.aq_inner.aq_parent
```

```
183             container.manage_delObjects([object.id])
184             # return a ObjectDeletedState() instance
185             return states.ObjectDeletedState()
186 
187         def _getObjectByUID(self, uid):
188             """
189             Searches a Object by UID in the plone reference_catalog.
190             If the Plone-Object could not be found, it will return None.
191 
192             @param uid:     UID of the object to search for
193             @type uid:      string
194             @return:        Plone-Object or None
195             """
196             return self.context.reference_catalog.lookupObject(uid)
197 
198         def _getObjectByPath(self, absolutePath):
199             """
200             Searches a Object by the absolute path of the object.
201             Path example: /data-fs/ploneSite/folder/object
202 
203             @param absolutePath:    Absolute path to the object to search for
204             @type absolutePath:     string
205             @return:                Plone-Object or None
206             """
207             # plone site root is not in catalog ...
208             portalObject = self.context.portal_url.getPortalObject()
209             portalPath = '/'.join(portalObject.getPhysicalPath())
210             if absolutePath==portalPath:
211                 # plone site root is searched, return it
212                 return portalObject
213             # for any other object we use the catalog tool
214             brains = self.context.portal_catalog({
215                     'path' : {
216                             'query' : absolutePath,
217                             'depth' : 0,
218                     },
219             })
220             if len(brains)==0:
221                 return None
222             else:
223                 return brains[0].getObject()
224 
225         def _findContainerObjectByPath(self, absoluteObjectPath):
226             """
227             Searches the Container of the currently not existing Plone-Object which
228             will be created.
229 
230             @param absoluteObjectPath:  Absolute path to the object (which does not
231                                         exist yet
232             @type absoluteObjectPath:   string
233             @return:                    Folderish Plone-Object or None
234             """
235             # get the path to the "parent" object
236             containerPath = os.path.dirname(absoluteObjectPath)
237             # search and return the "parent" object
238             return self._getObjectByPath(containerPath)
239 
240         def _getAbsolutePath(self, relativePath):
241             """
242             Converts a relative path (relative to Plone Site-Root) to a absolute
243             Path (containing the full URI from zope-Root object).
244             No path validation is done!
245             Example:
246                 relativePath :=     /folder/object
247                 absolutePath :=     /ploneSite/folder/object
248 
249             @param relativePath:    Any relative path
250             @type relativePath:     string
251             @return:                Absolute Path
252             @rtype:                 string
```

```
253             """
254             # get the portal path (path to plone site)
255             portalObject = self.context.portal_url.getPortalObject()
256             portalPath = '/'.join(portalObject.getPhysicalPath())
257             # concatinate with portalPath with relativePath
258             return portalPath + relativePath
259
260         def _updateProperties(self, object, properties):
261             """
262             Sets a list of properties on a object.
263             Warning: all currently set properties which are not in the
264             properties-list wille be removed!
265
266             @param object:      Plone-Object to set the properties on
267             @type object:       Plone-Object
268             @param properties:  list of propertes. See publisher.sender.extractor
269                                 for format details.
270             @param type:        list
271             @return:            None
272             """
273             self.logger.info('Updating properties (UID %s)' %
274                     (object.UID())
275             )
276             # we need to cleanup the properties. remove all properties
277             # from the object
278             propertiesToUpdateOrCreate = [p['id'] for p in properties]
279             currentProperties = object.propertyIds()
280             # delete old properties
281             propertiesToDelete = [id for id in currentProperties if id not
282                                                     in propertiesToUpdateOrCreate]
283             object.manage_delProperties(propertiesToDelete)
284             # get cleaned up list of properties
285             currentProperties = object.propertyIds()
286             # update or create properites
287             for prop in properties:
288                 if prop['id'] in currentProperties:
289                     # update property if existing ...
290                     object._updateProperty(
291                         id = prop['id'],
292                         value = prop['value'],
293                     )
294                 else:
295                     # ... otherwise
296                     object.manage_addProperty(
297                         id = prop['id'],
298                         value = prop['value'],
299                         type = prop['type'],
300                     )
301
302         def updateObjectPosition(self, object, metadata):
303             """
304             Updates the position of the object and it siblings (reset position of
305             all children of the parent object).
306             Objects, which do not exist at the sender instance, are moved to the
307             bottom.
308
309             @param metadata:        metadata dictionary containing UID, portal_type,
310                                     action, physicalPath and sibling_positions
311             @type metadata:         dict
312             @return:                None
313             """
314             positions = metadata['sibling_positions']
315             parent = object.aq_inner.aq_parent
316             object_ids = [o['id'] for o in parent._objects]
317
318             # move objects with no position info to the bottom
319             for id in object_ids:
320                 if id not in positions.keys():
321                     positions[id] = len(positions.keys())
322
```

```
323         # sort ids by positions
324         object_ids.sort(lambda a,b: cmp(positions[a], positions[b]))
325 
326         # order objects
327         parent.moveObjectsByDelta(object_ids, -len(object_ids))
328 
329         # reindex all objects
330         for id in object_ids:
331             parent[id].reindexObject()
332 
333 
334 
335 class TestConnection(BrowserView):
336     """
337     This BrowserView is used by the configlet of the module publisher.sender
338     to test a connection to the receiever.
339     """
340 
341     def __call__(self, *args, **kwargs):
342         """
343         Returns a 'ok' if the context is a Plone Site, otherwise
344         it returns 'Not a Plone-Site'.
345         @return:        Success message
346         @rtype:         string
347         """
348         if IPloneSiteRoot.providedBy(self.context):
349             return 'ok'
350         else:
351             return 'Not a Plone-Site'
```

Listing 14: decoder.py

```
1  # ... header wurde entfernt ...
2 
3  __author__ = """Jonas Baumann <j.baumann@4teamwork.ch>"""
4 
5  # global imports
6  import simplejson
7  import base64
8 
9  # plone imports
10 from Products.Archetypes.Field import FileField
11 from Products.Archetypes.Field import ImageField
12 from Products.Archetypes.Field import ReferenceField
13 
14 # publisher imports
15 from publisher.core import states
16 
17 class Decoder(object):
18     """
19     Decodes json data to dictionary and validates it.
20     It also validates and decodes all schema field values.
21     """
22 
23     def __init__(self, context):
24         """
25         Constructor: stores context as object attribute
26         @param context:         Plone object
27         @type:                  Plone Object
28         """
29         self.context = context
30 
31     def __call__(self, jsondata):
32         """
33         Decodes the jsondata to a dictionary, validates it,
34         unserializes the field values and returns it.
35         @return:        Data dictionary
36         @rtype:         dict
37         """
```

```
38          self.data = self.decodeJson(jsondata)
39          self.validate()
40          self.unserializeFields()
41          return self.data
42
43      def decodeJson(self, jsondata):
44          """
45          Decodes the JSON data with the simplejson module.
46          If the simplejson module cannot decode the string, a
47          DecodeError is raised.
48          @param jsondata:    JSON data
49          @type jsondata:     string
50          @return:            Decode Data dictionary
51          @rtype:             dict
52          @raise:             DecodeError
53          """
54          try:
55              data = simplejson.loads(jsondata)
56          except Exception, e:
57              raise states.DecodeError(str(e))
58          return data
59
60      def validate(self):
61          """
62          Validates, if all required values are provided. If a
63          error occures, a PartialError is raised.
64          @raise:             PartialError
65          @return:            None
66          """
67          structure = {
68              'fielddata' : [],
69              'metadata' : [
70                  'UID',
71                  'portal_type',
72                  'action',
73                  'physicalPath',
74                  'sibling_positions',
75              ]
76          }
77          for key in structure.keys():
78              if key not in self.data.keys():
79                  raise states.PartialError('Missing "%s"' % key)
80              else:
81                  for subkey in structure[key]:
82                      if subkey not in self.data[key]:
83                          raise states.PartialError('Missing "%s.%s"' % (key, subkey))
84
85      def getSchema(self):
86          """
87          Returns the Schema of the portal_type defined in the metadata.
88          @return:        Archetypes Schema object
89          @rtype:         Schema
90          """
91          typename = self.data['metadata']['portal_type']
92          types = self.context.archetype_tool.listRegisteredTypes()
93          typeclass = filter(lambda x:x['portal_type']==typename, types)[0]['klass']
94          return typeclass.schema
95
96      def unserializeFields(self):
97          """
98          Unserializes the fielddata and optimizes it of the modifiers of
99          the fields.
100         Gets and sets from / to **self.data**
101         """
102         schema = self.getSchema()
103         for field in schema.fields():
104             name = field.getName()
105             if name not in self.data['fielddata'].keys():
106                 continue
107             # DateTimeField doesnt need to be converted t DateTime
```

```
108             # FileFields are base64 encoded
109             if isinstance(field, FileField):
110                 value = self.data['fielddata'][name]
111                 if isinstance(value, dict):
112                     # decode it
113                     data = base64.decodestring(value['data'])
114                     filename = value['filename']
115                     # process it
116                     file, mimetype, filename = field._process_input(data, filename=filename
                            )
117                     # we only use the file object
118                     self.data['fielddata'][name] = file
119             # ReferenceField: remove bad UIDs
120             if isinstance(field, ReferenceField):
121                 cleaned = []
122                 for uid in self.data['fielddata'][name]:
123                     obj = self.context.reference_catalog.lookupObject(uid)
124                     if obj:
125                         cleaned.append(uid)
126                 self.data['fielddata'][name] = cleaned
127             # ImageField: treat empty files special
128             if isinstance(field, ImageField):
129                 if len(self.data['fielddata'][name])==0:
130                     self.data['fielddata'][name] = 'DELETE_IMAGE'
131             # FileField (direct): treat empty files special
132             if field.__class__==FileField:
133                 if len(self.data['fielddata'][name])==0:
134                     self.data['fielddata'][name] = 'DELETE_FILE'
```

Modul publisher.sender

Listing 15: views.py

```
1  # ... header wurde entfernt ...
2
3  __author__ = """Jonas Baumann <j.baumann@4teamwork.ch>"""
4
5  # python imports
6  from StringIO import StringIO
7  import logging
8
9  # zope imports
10 from Products.Five import BrowserView
11
12 # plone imports
13 from Products.CMFPlone.interfaces import IPloneSiteRoot
14 from Products.statusmessages.interfaces import IStatusMessage
15
16 # publisher imports
17 from publisher.sender.persistence import Queue, Config
18 from publisher.sender.utils import sendJsonToRealm
19 from publisher.sender import extractor
20 from publisher.sender import getLogger
21 from publisher.core import states
22
23 """
24 @var BATCH_SIZE:        Maximum amount of Jobs to be performed at one ExecuteQueue call
25 """
26 BATCH_SIZE = 10
27
28 class PublishObject(BrowserView):
29     """
30     This BrowserView adds the current object (self.context) to the publishing queue.
31     """
32
33     def __call__(self, *args, **kwargs):
34         """
```

```
35          The __call__ method is used to execute the BrowserView. It creates and
36          adds a "PUSH"-Job on the current context to the queue.
37          @param args:    list of unnamed arguments
38          @type args:     list
39          @param kwargs:  dict of named keyword-arguments
40          @type kwargs:   dict
41          @return:        Redirect to object`s default view
42          """
43          self.logger = getLogger()
44          # This View should not be executed at the PloneSiteRoot
45          if IPloneSiteRoot.providedBy(self.context):
46              raise Exception('Not allowed on PloneSiteRoot')
47          # get username
48          user = self.context.portal_membership.getAuthenticatedMember()
49          username = user.getUserName()
50          # create Job
51          queue = Queue(self.context)
52          queue.createJob('push', self.context, username)
53          self.logger.info('Created "%s" Job for "%s" at %s' % (
54                  'push',
55                  self.context.Title(),
56                  '/'.join(self.context.getPhysicalPath()),
57          ))
58          # status message
59          IStatusMessage(self.request).addStatusMessage(
60                  'This object has been added to the queue.',
61                  type='info'
62          )
63          return self.request.RESPONSE.redirect('./view')
64
65
66  class DeleteObject(BrowserView):
67      """
68      Add a object to the queue with the action "delete".
69      """
70
71      def __call__(self, *args, **kwargs):
72          """
73          Add the current context as delete-job to the queue, creates a status
74          message to inform the user and returns to the default view.
75          @param args:    list of unnamed arguments
76          @type args:     list
77          @param kwargs:  dict of named keyword-arguments
78          @type kwargs:   dict
79          @return:        Redirect to object`s default view
80          """
81          self.logger = getLogger()
82          # This view should not be executed at the PloneSiteRoot
83          if IPloneSiteRoot.providedBy(self.context):
84              raise Exception('Not allowed on PloneSiteRoot')
85          # get username
86          user = self.context.portal_membership.getAuthenticatedMember()
87          username = user.getUserName()
88          # create Job
89          queue = Queue(self.context)
90          queue.createJob('delete', self.context, username)
91          self.logger.info('Created "%s" Job for "%s" at %s' % (
92                  'delete',
93                  self.context.Title(),
94                  '/'.join(self.context.getPhysicalPath()),
95          ))
96          # status message
97          IStatusMessage(self.request).addStatusMessage(
98                  'This object will be deleted at the remote sites.',
99                  type='info'
100         )
101         return self.request.RESPONSE.redirect('./view')
102
103
104
```

```
105 class ExecuteQueue(BrowserView):
106     """
107     Executes the Queue and sends BATCH_SIZE amount of Jobs to the target realms.
108     """
109
110     def __call__(self, *args, **kwargs):
111         """
112         Handles logging purposes and calls execute() method.
113         @param args:    list of unnamed arguments
114         @type args:     list
115         @param kwargs:  dict of named keyword-arguments
116         @type kwargs:   dict
117         @return:        Redirect to object`s default view
118         """
119         self.logger = getLogger()
120         # register our own logging handler for returning logs afterwards
121         logStream = StringIO()
122         logHandler = logging.StreamHandler(logStream)
123         self.logger.addHandler(logHandler)
124         # be sure to remove the handler!
125         try:
126             # get config and queue
127             self.config = Config(self.context)
128             self.queue = Queue(self.context)
129             # execute queue
130             self.execute()
131         except:
132             self.logger.removeHandler(logHandler)
133             # re-raise exception
134             raise
135         # get logs
136         self.logger.removeHandler(logHandler)
137         logStream.seek(0)
138         log = logStream.read()
139         del logStream
140         del logHandler
141         return log
142
143     def getActiveRealms(self):
144         """
145         @return: a list of active Realms
146         @rtype: list
147         """
148         if '_activeRealms' not in dir(self):
149             self._activeRealms = [r for r in self.config.getRealms() if r.active]
150         return self._activeRealms
151
152     def execute(self):
153         """
154         Executes the jobs from the queue. Maximum amount of performed jobs can
155         be set with the BATCH_SIZE global.
156         @return: None
157         """
158         # jobCounter counts the amount of executed jobs
159         jobCounter = 0
160         jobs = self.queue.countJobs()
161         self.logger.info('Executing Queue: %i of %i objects to %i realms' % (
162                 jobs>BATCH_SIZE and BATCH_SIZE or jobs,
163                 self.queue.countJobs(),
164                 len(self.getActiveRealms()),
165         ))
166         while self.queue.countJobs()>0 and (BATCH_SIZE<1 or jobCounter<BATCH_SIZE):
167             # get job from queue
168             job = self.queue.popJob()
169             # execute job
170             self.executeJob(job)
171             # remove cache file from file system / delete job
172             job.removeJob()
173             jobCounter += 1
174
```

```
175     def executeJob(self, job):
176         """
177         Executes a Job: sends the job to all available realms.
178         @param job:     Job object to execute
179         @type job:      Job
180         """
181         # get data from chache file
182         json = job.getData()
183         self.logger.info('-' * 100)
184         self.logger.info('executing "%s" on "%s" (at %s | UID %s)' % (
185                 job.action,
186                 job.objectTitle,
187                 job.objectPath,
188                 job.objectUID,
189         ))
190         self.logger.info('... request data length: %i' % len(json))
191         for realm in self.getActiveRealms():
192             self.logger.info('... to realm %s' % (
193                     realm.url,
194             ))
195             # send data to each realm
196             state = sendJsonToRealm(json, realm, 'publisher.receive')
197             if isinstance(state, states.ErrorState):
198                 self.logger.error('... got result: %s' % state.toString())
199             else:
200                 self.logger.info('... got result: %s' % state.toString())
```

Listing 16: extractor.py

```
1   # ... header wurde entfernt ...
2
3   __author__ = """Jonas Baumann <j.baumann@4teamwork.ch>"""
4
5   # global imports
6   import simplejson
7   import base64
8
9   # zope imports
10  from OFS.Image import File
11
12  # plone imports
13  from Products.Archetypes.Field import DateTimeField
14  from Products.Archetypes.Field import FileField
15
16
17  class Extractor(object):
18      """
19      The Extractor module is used for extracting the data from a Object and
20      pack it with json.
21      """
22
23      def __call__(self, object, action):
24          """
25          Extracts the required data (action dependent) from a object for
26          creating a Job.
27          @param object:      Plone Object to export data from
28          @param action:      Action to perform [push|delete]
29          @type action:       string
30          @return:            data (json "encoded")
31          @rtype:             string
32          """
33          self.object = object
34          data = {}
35          if action=='delete':
36              # we dont need big data for deleting ;)
37              data['fielddata'] = {}
38              data['properties'] = []
39          else:
40              data['fielddata'] = self.getFieldData()
```

```
41              data['properties'] = self.getPropertyData()
42          data['metadata'] = self.getMetadata(action)
43          # convert to json
44          jsondata = self.convertToJson(data)
45          return jsondata
46
47      def getFieldData(self):
48          """
49          Extracts data from the object fields and creates / returns a dictionary with
50          the data. Objects are converted to string.
51          @return:    dictionary with extracetd data
52          @rtype:     dict
53          """
54          data = {}
55          schema = self.object.schema
56          for field in schema.fields():
57              name = field.getName()
58              value = field.getRaw(self.object)
59              value = self.fieldSerialization(field, value)
60              data[name] = value
61          return data
62
63      def fieldSerialization(self, field, value):
64          """
65          Custom serialization for fields which provide field values that are incompatible
66          with simplejson / JSON-standard.
67          @param field:   Field-Object from Schema
68          @type field:    Field
69          @param value:   Return-Value of the Raw-Accessor of the Field on the current
                context
70          @type value:    string or stream
71          @return:        JSON-optimized value
72          @rtype:         string
73          """
74          # DateField : returns a DateTime-Object as value. We cast it to string and it
75          #   looks like '2010-05-31 13:06:01.925652'
76          if isinstance(field, DateTimeField) and value:
77              value = str(value)
78          # FileField : returns a File-Object, but TextField is a FileField too, so we
79          # have to detect the type of value. Binary data must be encoded with base64
80          elif isinstance(field, FileField):
81              if isinstance(value, File):
82                  value = {
83                          'filename' : value.filename,
84                          'data' : base64.encodestring(value.data),
85                  }
86          return value
87
88      def getMetadata(self, action):
89          """
90          Returns a dictionary with metadata about this object. It contains also the action.
91          @param action:  publishing action [push|delete]
92          @type action:   string
93          @return:        metadata dictionary
94          @rtype:         dict
95          """
96          parent = self.object.aq_inner.aq_parent
97          # get object positions
98          positions = {}
99          for obj_id in parent.objectIds():
100             positions[obj_id] = parent.getObjectPosition(obj_id)
101         # create metadata dict
102         data = {
103             'UID' : self.object.UID(),
104             'portal_type' : self.object.portal_type,
105             'action' : action,
106             'physicalPath' : self.getRelativePath(),
107             'sibling_positions' : positions,
108         }
109         return data
```

```
110
111     def getPropertyData(self):
112         """
113         Returns a list of dictonaries each representing a property.
114         Example Return: [
115             {
116                 'type' : 'string',
117                 'id' : 'title',
118                 'value' : 'test1',
119                 'mode' : 'wd',
120             },
121             {
122                 'type' : 'text',
123                 'id' : 'blubb',
124                 'value' : 'asdfsadf
125 asdf',
126             },
127         ]
128
129         @return:    list of properties
130         @rtype:     list
131         """
132         properties = []
133         for prop in self.object._propertyMap():
134             # create a copy (we dont want to change the effective property)
135             prop = prop.copy()
136             # add the value
137             prop['value'] = self.object.getProperty(prop['id'])
138             properties.append(prop)
139         return properties
140
141     def getRelativePath(self):
142         """
143         Returns the relative path (relative to plone site) to the current context object.
144         @return:    relative path
145         @rtype:     string
146         """
147         path = '/'.join(self.object.getPhysicalPath())
148         portalPath = '/'.join(self.object.portal_url.getPortalObject().getPhysicalPath())
149         if not path.startswith(portalPath):
150             raise TypeError('Expected object to be in a portal object -.-')
151         return path[len(portalPath):]
152
153     def convertToJson(self, data):
154         """
155         Converts a dictionary to a JSON-string
156         @param data:    data dictionary
157         @type data:     dict
158         @return:        JSON
159         @rtype:         string
160         """
161         return simplejson.dumps(data, sort_keys=True)
```

Listing 17: persistence.py

```
1  # ... header wurde entfernt ...
2
3  __author__ = """Jonas Baumann <j.baumann@4teamwork.ch>"""
4
5  # Python imports
6  import os
7  import time
8
9  # Zope imports
10 from Acquisition import aq_inner
11 from persistent import Persistent
12 from persistent.list import PersistentList
13 from zope import interface, component
14 from zope.annotation.interfaces import IAnnotations
```

```
15
16  # Plone imports
17  from Products.CMFCore.utils import getToolByName
18  from Products.CMFPlone.interfaces import IPloneSiteRoot
19
20  # publisher imports
21  from interfaces import IConfig, IQueue
22  from publisher.sender import extractor
23
24  class Config(object):
25      """
26      The Config object is registered via zcml as adapter. It stores the
27      configured realms
28      """
29      interface.implements(IConfig)
30      component.adapts(IPloneSiteRoot)
31
32      def __init__(self, context):
33          """
34          Constructor: load the annotations, which are stored on the
35          plone site.
36
37          @param context:     any context
38          @type context:      Plone object
39          """
40          # get plone site
41          self.context = aq_inner(context.portal_url.getPortalObject())
42          # get annotations for plone site
43          self.annotations = IAnnotations(self.context)
44
45      def getRealms(self):
46          """
47          Returns a PersistentList of Realm objects
48          @return:    Realm objects
49          @rtype:     PersistentList
50          """
51          return self.annotations.get('publisher-realms', PersistentList())
52
53      def _setRealms(self, list):
54          """
55          Stores a PersistentList of Realm objects
56          @param list:    Realm objects
57          @type list:     PersistentList
58          @return:        None
59          """
60          if not isinstance(list, PersistentList):
61              raise TypeError('Excpected PersistentList')
62          self.annotations['publisher-realms'] = list
63
64      def appendRealm(self, realm):
65          """
66          Appends a Realm to the realm list
67          @param realm:   Realm object
68          @type realm:    Realm
69          @return:        None
70          """
71          if not isinstance(realm, Realm):
72              raise TypeError('Excpected Realm object')
73          list = self.getRealms()
74          list.append(realm)
75          self._setRealms(list)
76
77      def removeRealm(self, realm):
78          """
79          Removes a Realm from the realm list
80          @param realm:   Realm object
81          @type realm:    Realm
82          @return:        None
83          """
84          if not isinstance(realm, Realm):
```

```
85              raise TypeError('Excpected Realm object')
86          list = self.getRealms()
87          list.remove(realm)
88          self._setRealms(list)
89
90      def getDataFolder(self):
91          """
92          Returns the path to the data folder. If it does not exist, it will
93          be created.
94          @return:        absolute file system path
95          @rtype:         string
96          """
97          path = self.annotations.get('publisher-dataFolder', None)
98          if not path:
99              path = os.path.join(os.environ['INSTANCE_HOME'], 'var', 'publisher')
100             self.setDataFolder(path)
101         # create if not existing
102         if not os.path.exists(path):
103             os.mkdir(path)
104         return path
105
106     def setDataFolder(self, path):
107         """
108         Sets the data folder path
109         @param path:    absoliute path to the data folder
110         @type path:     string
111         @return:        None
112         """
113         self.annotations['publisher-dataFolder'] = path
114
115 class Queue(object):
116     """
117     The Queue adapter stores a list of Jobs to process.
118     """
119     interface.implements(IQueue)
120     component.adapter(IPloneSiteRoot)
121
122     def __init__(self, context):
123         """
124         Constructor: load the annotations, which are stored on the
125         plone site.
126
127         @param context:     any context
128         @type context:      Plone object
129         """
130         self.context = aq_inner(context.portal_url.getPortalObject())
131         self.annotations = IAnnotations(self.context)
132
133     def getJobs(self):
134         """
135         Returns a PersistentList of Job objects
136         @return:        job-objects
137         @rtype:         PersistentList
138         """
139         return self.annotations.get('publisher-queue', PersistentList())
140
141     def _setJobs(self, list):
142         """
143         Stores a PersistentList of Job objects
144         @param list:    list of jobs
145         @type list:     PersistentList
146         @return:        None
147         """
148         if not isinstance(list, PersistentList):
149             raise TypeError('Excpected PersistentList')
150         self.annotations['publisher-queue'] = list
151
152     def appendJob(self, job):
153         """
154         Appends a Job to the queue
```

```
155         @param job:     Job object
156         @type:          Job
157         @return:        None
158         """
159         if not isinstance(job, Job):
160             raise TypeError('Excpected Job object')
161         list = self.getJobs()
162         list.append(job)
163         self._setJobs(list)
164 
165     def createJob(self, *args, **kwargs):
166         """
167         Creates a new Job object, adds it to the queue
168         and returns it.
169         Arguments are redirected to the Job-Constructor.
170         @return:    Job object
171         @rtype:     Job
172         """
173         job = Job(*args, **kwargs)
174         self.appendJob(job)
175         return job
176 
177     def removeJob(self, job):
178         """
179         Removes a Job from the queue
180         @param job:     Job object
181         @type job:      Job
182         @return:        None
183         """
184         if not isinstance(job, Job):
185             raise TypeError('Excpected Job object')
186         list = self.getJobs()
187         list.remove(job)
188         self._setJobs(list)
189 
190     def countJobs(self):
191         """
192         Returns the amount of jobs in the queue.
193         Used in combination with popJob()
194         @return:        Amount of jobs in the queue
195         @rtype:         int
196         """
197         return len(self.getJobs())
198 
199     def popJob(self):
200         """
201         Returns the oldest Job from the queue. The Job will be
202         removed from the queue immediately!
203         @return:        Oldest Job object
204         @rtype:         Job
205         """
206         return self.getJobs().pop(0)
207 
208 
209 class Job(Persistent):
210     """
211     A Job object contains action, object and the user who triggered the job.
212     It is stored in the Queue and is executed asynchronous.
213     """
214 
215     def __init__(self, action, object, username):
216         """
217         Constructor: sets the given arguments.
218         @param action:      action type [push|delete]
219         @type action:       string
220         @param object:      plone object to run job on
221         @type object:       Plone object
222         @param username:    Name of the user which performed the action
223         @type username:     string
224         """
```

```
225             super(Persistent, self).__init__()
226             self.action = action
227             self.username = username
228             self.objectUID = object.UID()
229             self.objectPath = '/'.join(object.getPhysicalPath())
230             self.objectTitle = object.Title()
231             self._extractData(object)
232 
233         def _extractData(self, object):
234             """
235             Extracts the data from the object and stores the JSON string
236             in a cache-file.
237             @param object:      plone object to run job on
238             @type object:       Plone object
239             """
240             # create new data file
241             dir = Config(object).getDataFolder()
242             i = 1
243             file = None
244             while not file:
245                 filename = '%s.%s.%s.json' % (
246                     object.UID(),
247                     time.strftime('%Y%m%d-%H%M%S'),
248                     str(i).rjust(3, '0')
249                 )
250                 file = os.path.join(dir, filename)
251                 if os.path.exists(file):
252                     file = None
253             f = open(file, 'w')
254             # extract data
255             data = extractor.Extractor()(object, self.action)
256             # write data
257             f.write(data)
258             f.close()
259             self.dataFile = file
260 
261         def getData(self):
262             """
263             Loads the JSON-data from the cache file and returns
264             the JSON-string
265             @return:    JSON data
266             @rtype:     string
267             """
268             f = open(self.dataFile)
269             data = f.read()
270             f.close()
271             return data
272 
273         def removeJob(self):
274             """
275             Removes the cache file for this job from the file system
276             """
277             os.remove(self.dataFile)
278 
279         def getObject(self, context):
280             """
281             Returns the object with UID stored in this Job. This method
282             requires any context for getting the reference_catalog.
283             @param context:     Any Plone object
284             @type context:      Plone Object
285             @return:            Context object of this Job
286             @rtype:             Plone object
287             """
288             reference_tool = getToolByName(context, 'reference_catalog')
289             return reference_tool.lookupObject(self.objectUID)
290 
291 
292     class Realm(Persistent):
293         """
294         A Realm object provides information about a target plone instance (receiver)
```

```
295     which should have installed publisher.receiver.
296     It stores and provides information such as URL or credentials.
297     URL+username should be unique!
298     """
299
300     active = 0
301     url = ''
302     username = ''
303     password = ''
304
305     def __init__(self, active, url, username, password):
306         """
307         Constructor: stores the given arguments in the object
308         @param active:      Is this realm active?
309         @type active:       boolean or int
310         @param url:         URL to the plone site of the target realm
311         @type url:          string
312         @param username:    Username of the user to publish data with on this realm
313         @type username:     string
314         @param password:    Password of the User with **username**
315         @type password:     string
316         """
317         self.active = active and 1 or 0
318         self.url = url
319         self.username = username
320         self.password = password
```

Listing 18: utils.py

```
1   # ... header wurde entfernt ...
2
3   __author__ = """Jonas Baumann <j.baumann@4teamwork.ch>"""
4
5   # global imports
6   import os.path
7   import urllib, urllib2, base64
8
9   # publisher imports
10  from publisher.sender.persistence import Realm
11  from publisher.core import communication
12
13  def sendJsonToRealm(json, realm, serverAction):
14      """
15      Sends the json data to a realm with the given serverAction and
16      parses the response.
17      @param json:            JSON data
18      @type json:             string
19      @param realm:           Realm object of the target instance
20      @type realm:            Realm
21      @param serverAction:    Name of the BrowserView on the target instance
22      @type serverAction:     string
23      @return:                A CommunicationState
24      @rtype:                 CommunicationState
25      """
26      if not isinstance(realm, Realm):
27          TypeError('Excpected Realm instance')
28      data = {'jsondata' : json}
29      html = sendRequestToRealm(data, realm, serverAction)
30      return communication.parseResponse(html)
31
32  def sendRequestToRealm(data, realm, serverAction):
33      """
34      Makes a HTTP-Request to a realm and sends the given data on
35      the provided serverAction
36      @param data:            dictionary of parameters to send within the HTTP request
37      @type data:             dict
38      @param realm:           Realm object of the target instance
39      @type realm:            Realm
40      @param serverAction:    Name of the BrowserView on the target instance
```

```
41      @type serverAction:     string
42      @return:                Response Text
43      @rtype:                 string
44      """
45      if not isinstance(realm, Realm):
46          TypeError('Excpected Realm instance')
47      url = os.path.join(realm.url, serverAction)
48      credentials = ':'.join([realm.username.encode('hex'), realm.password.encode('hex')])
49      credentials = str(base64.encodestring(credentials)).strip()
50      headers = {
51              'User-Agent'            : 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)',
52              'Content-Type'          : 'application/x-www-form-urlencoded',
53              'Cookie'                : '__ac=' + credentials,
54      }
55      request = urllib2.Request(url, urllib.urlencode(data), headers)
56      response = urllib2.urlopen(request)
57      return response.read()
```