RSS Integration via Java EE

Paul Schalow

20. August 2010

Humboldt-Universität zu Berlin, Institut für Wirtschaftsinformatik

Kurzfassung

Wie lassen sich abonnierte RSS Feeds auf verschiedenen Clients synchron halten? Eine Möglichkeit der bloßen Betrachtung besteht in der Verwendung von Offline-Clients (z.B. Microsoft Outlook). Nachteilig hierbei zu bewerten ist, dass bei einer Nutzung von mehreren Clients (z.B. PCs) keine zentrale Synchronisation nach aktuellem Stand möglich ist, wie es beispielsweise über IMAP für E-Mails realisiert ist. Somit würde jeder Client für sich sämtliche Nachrichten eingestellter RSS-Feeds aggregieren und als ungelesen markieren. Bisherige Umsetzungen der Synchronisation erweisen sich als sehr unkomfortabel in der Nutzung und schlecht erweiterbar. Diese Arbeit diskutiert verschiedene Lösungsansätze. Es wird ein Weg aufgezeigt, wie das Problem gelöst werden kann, indem eine Java EE Middleware entwickelt wird, welche über IMAP mit Clients kommuniziert. Dadurch können Vorteile generiert werden, die teils auf die Nutzung von Java EE teils auf den gewählten Lösungsansatz zurückzuführen sind.

Abkürzungen

API Application Program Interface

JAR Java Archive

JDK Java Development Kit

JRE Java Runtime Environment

JEE Java EE; Java Enterprise Edition

J2EE obsolete Abkürzung für JEE

EAR Enterprise Archive

EIS Enterprise Information System

ERP Enterprise Resource Planning

IMAP Internet Message Access Protocol

MDB Message Driven Bean

MVC Model View Control

RAR Resource Adapter Archive

RSS Really Simple Syndication

Inhaltsverzeichnis

Listing 1: Beispiel RSS Feed

```
<rss version="0.91">
  <channel>
    <title>Sample RSS Feed</title>
    <link>http://localhost/</link>
    <description>This feed represents a simple sample.
      </description>
    <language>en-us</language>

    <item>
      <title>A sample message</title>
      <link>http://localhost/rss/sample/0</link>
      <description>RSS Feeds are used to ...
        </description>
    </item>
  </channel>
</rss>
```

1 Motivation

RSS (Really Simple Syndication) ist ein flexibles Format, welches für den Austausch strukturierter und dynamischer Daten wie beispielsweise Schlagzeilen, Blogs oder Wiki Änderungen genutzt werden kann. Es basiert auf XML und stellt ein, aus der Perspektive des Clients (Nutzers), nur lesbares Protokoll dar.[15] Das Listing 1 repräsentiert ein beispielhaften RSS Feed.

Es besteht die Möglichkeit, RSS beispielsweise mittels eines Online Aggregators zu abonnieren und zu lesen oder indem ein Client (eine Applikation) eingesetzt wird (z.B. Outlook). Nach dem aktuellen Stand entstehen bei der Nutzung auf verschiedenen Rechner eine Anzahl Probleme. Diese resultieren aus der mangelnden Synchronisationsfähigkeit von RSS. Angenommen es werden zwei Clients (zwei Rechner) benutzt, auf denen jeweils Outlook verwendet wird um den selben RSS Feed zu abonnieren. Beide Instanzen funktionieren unabhängig voneinander. Das bedeutet, dass wenn Outlook A verwendet wird, um die neuesten RSS Nachrichten zu lesen, dann bekommt Outlook B davon nichts mit. Dementsprechend würden bei der nächsten Nutzung von Outlook B sämtliche Nachrichten, die bereits auf Outlook A gelesen wurden auch als ungelesen markiert. Ein Online-Aggregator als zentraler Service kann dieses Problem beheben. Doch, wenn beispielsweise ein Client auch benutzt werden soll, wenn mal keine Internetverbindung besteht, ist diese Lösung hinfällig. Oder, was wenn einerseits auf den heimischen Rechnern der präferierte RSS Client nutzbar sein, andererseits auch spontan ein Service im Internetcafe ermöglicht werden soll – und dennoch der Status, ob eine Nachricht z.B. bereits gelesen wurde, nicht verloren gehen darf. Lässt sich eine Möglichkeit schaffen, bei der sämtliche Kriterien erfüllt werden? Die Nutzung von RSS soll dementsprechend über verschiedene

Clients integriert werden.

In den folgenden Kapiteln wird eine Lösung umgesetzt und dafür folgende Fragen beantwortet: Welcher Ansatz ermöglicht es RSS auf verschiedenen Rechnern synchron zu halten? Wie lässt sich ein fremdes Protokoll (in diesem Fall IMAP) mit Java EE integrieren?

2 Anforderungen und Einordnung in bestehende Projekte

Im Folgenden werden die Anforderungen, die ein potentielles Produkt abdecken soll, aufgestellt. Diese werden auf vorhandene oder potentielle Umsetzungen abgebildet. Darauf basierend wird eine *Make Or Buy* Entscheidung getroffen.

Anforderungen

❶ **Synchronisation** RSS Feeds werden auf allen Clients synchron gehalten.

❷ **Offlinefähigkeit** Eine Betrachtung bereits abgerufener Nachrichten ist auch unabhängig von einem Internetzugang möglich.

❸ **Outlook-Kompatibilität**

❹ **Client-Unabhängigkeit**

❺ **Änderungen** an bereits abgerufenen Nachrichten werden registriert und dem Nutzer abgeändert dargestellt (und nicht etwa als zusätzliche neue Nachricht).

Vorhandene und potentielle Realisierungen

- RSS Synchronisation zwischen Outlook: Das Tool *Easy2Sync für Outlook [1]* synchronisiert mehrere Outlook Instanzen.
- Lieferung per Mail nach IMAP Postfach: Das Tool *rss2imap* [7] fragt RSS Feeds ab und sendet die Nachrichten an IMAP Server als E-Mail-Nachrichten. Somit kann der Betrachter gelieferte RSS Nachrichten auf dem IMAP Server beispielsweise als *gelesen* markieren.
- Spezieller Client: Es ließe sich eine Client-Anwendung entwickeln, die als RSS Reader fungiert und sich synchronisiert.
- Online Aggregator: Bloglines [8] stellt eine Website dar, auf der individuell RSS Feeds zusammengestellt, abonniert und gelesen werden können. Indem ein individueller Online Aggregator in Eigenentwicklung erarbeitet werden würde, wären zusätzliche Features integrierbar.
- Proprietäres (z.B. Outlook) Plug-In: Potentielles Tool, welches sich vollständig in die verschiedenen RSS Clients integriert und RSS Nachrichten synchronisiert.

Tabelle 1: Evaluierung existierender oder potentieller Umsetzungen anhand spezifischer Anforderungen

	❶	❷	❸	❹	❺
RSS Synchronisation zwischen Outlook	•	•	•	◦	•
Lieferung per Mail nach IMAP Postfach	•	•	•	•	◦
Spezieller Client	•	•	◦	◦	•
Online Aggregator (Website)	•	◦	◦	•	•
Proprietäres (z.B. Outlook) Plug-In	•	•	•	◦	•
Spezielle Middleware	•	•	•	•	•

- Spezielle Middleware: Eine Applikation, die direkt als Vermittler zwischen RSS Feeds und Betrachter fungiert. Der Betrachter kann dabei entweder direkt ein Nutzer sein oder ein für die Betrachtung von RSS Feeds ausgerichtetes Programm (z.B. Outlook).

Tabelle 1 illustriert den Zusammenhang verschiedener Lösungsansätze und welche Anforderungen durch jene abgedeckt werden (• Feature wird unterstützt; ◦ Feature wird nicht unterstützt).

Einzig eine „spezielle Middleware“ erfüllt sämtliche Anforderungen. Damit wurde die Notwendigkeit einer Neuentwicklung demonstriert. Wie in Abschnitt 4.1.2 erläutert wird, existieren zwar bereits implementierte Lösungen dieser Art. Diese decken jedoch nur ansatzweise die genannten Anforderungen ab.

3 Theoretischer Background

In diesem Kapitel werden die für das Projekt notwendigen theoretischen Grundlagen erarbeitet.

Java EE (auch JEE; ehemals J2EE) kürzt „Java Enterprise Edition“ ab. Es basiert auf der Java SE Plattform.

Java EE bietet eine API und Lauftzeitumgebung um mehrschichtige, skalierbare, verfügbare und sichere Netzwerkapplikationen (Enterprise Systeme) zu entwickeln und auszuführen respektive Informationssysteme (IS) miteinander zu integrieren.[11, S.13f]

3.1 Java EE Schichten

Mehrschichtig bedeutet, dass beispielsweise folgende Schichten miteinerander interagieren [11, S.15ff]:

Clientschicht

Webschicht besteht aus Komponenten, die das Zusammenspiel von Clients und Businesschicht koordinieren.

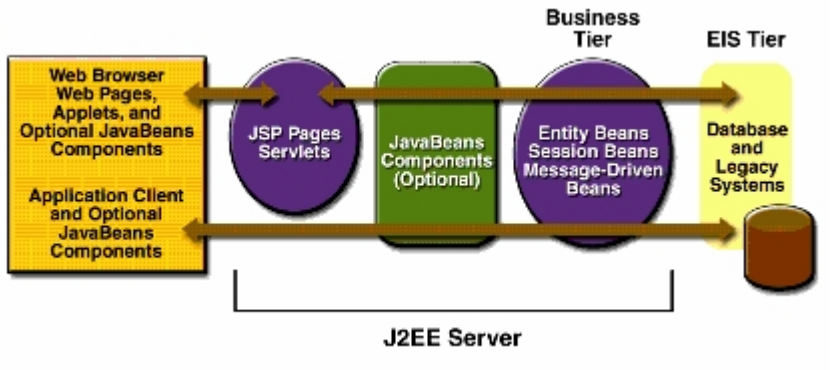

Abbildung 1: Beispielschichten einer Java EE Applikation [5, S.7]

Businessschicht (Business Tier) besteht aus Komponenten, die die Geschäftslogik einer Anwendung bereitstellen. Dafür werden beispielsweise folgende Java EE Technologien eingesetzt (diese werden im späteren Verlauf erläutert):

- Enterprise JavaBeans Komponenten,
- Java Persistence API *Entitäten.*

Enterprise Information System Schicht Die EIS besteht aus Datenbank-Servern, ERP Systemen und anderen Datenquellen. Beim Zugriff auf die EIS Schicht kommen beispielsweise folgende Java EE Technologien zum Einsatz:

- Java Persistence API (JPA),
- Java EE Connector Architecture.

Abbildung 1 illustriert potentielle Schichten einer Java EE Applikation.

3.2 Java EE Container

Eine Java EE Applikation besteht aus einem oder mehreren Containern. Diese beinhalten entweder wiederum Container (wie das EAR) oder bestimmten Code. Container werden in einem Application Server „deployed“ und von diesem verwaltet (managed).[5, S.8ff]

Abbildung 2 illustriert einige potentielle Containertypen, die im Folgenden erläutert werden[5, S.10]:

Java EE Server Die Laufzeitumgebung eines Java EE Produkts. Entweder wird ein Java EE Server so konfiguriert, dass er EJB und Web Container individuell ausführt oder aber ihm wird ein Enterprise Archive (EAR) „deployed“, welches aus Web oder EJB Containern bestehen kann.

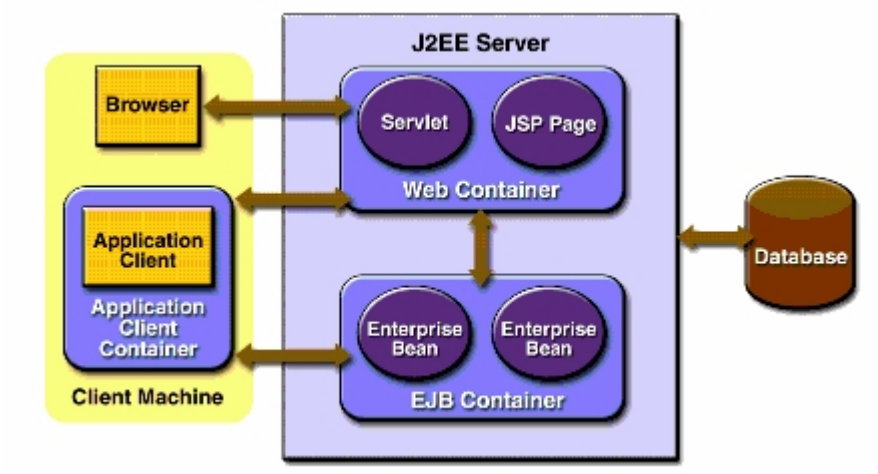

Abbildung 2: Beispielhafte Container einer Java EE Applikation [5, S.10]

Enterprise JavaBeans (EJB) Container verwaltet die Ausführung von Enterprise Beans für Java EE Applikationen. Diese werden in Java Archiven (JAR) gepackt und durch den Java EE Server ausgeführt.

Web Container verwaltet die Ausführung beispielsweise von Java ServerPages (JSP) oder Servlet Kompontenen in Web Archiven (WAR). Zudem dürfen diese Container auch EJBs enthalten. Sie werden durch den Java EE Server ausgeführt.

Application Client Container Verwaltet die Ausführung von Application Client Komponenten. Application Clients und deren Container werden auf der Client Seite ausgeführt.

Resource Adapter Container Wird im nächsten Abschnitt erläutert. Resource Adapter werden in Resource Adapter Archiven (RAR) gepackt.

3.3 Von Java EE bereitsgestellte und im Projekt verwendete APIs

Servlets Java Klassen, die in der Webschicht platziert sind sowie dynamisch Requests (HTTP) bearbeiten und Antworten (Responses) generieren. Sie repräsentieren üblicherweise HTML Seiten.[11, S.16]

EJBs Java Klassen, die die Geschäftslogik einer Enterprise Anwendung kapseln und in einem EJB Container gepackt sind. Es existieren drei Arten:

- Session Beans,
- Entity Beans und

- Message Driven Beans (MDB).

  Session Beans repräsentieren eine vorübergehende Konversation mit einem Client, dessen flüchtig gespeicherte Daten nach Beendigung nicht mehr zugreifbar sind. Entity Beans persistieren Daten in einer Datenbank-Zeile. Message Driven Beans erweitern Session Beans über den Java Message Service (JMS) *message listener*, womit eine lose Kopplung und eine asynchrone Abarbeitung von gesendeten Nachrichten möglich wird.[5, S.7f]

Java Connector Architecture (JCA) Systeme müssen teilweise mit anderen Systemen integriert werden. Dafür wird bei Java EE die JCA bereitgestellt.

Exkurs: Hypothetisch soll eine Java EE Applikation auf einem bestimmten Port mit einem bestimmten Protokoll (nicht HTTP) kommunizierbar sein. Es wird angenommen, dass in diesem Fall die äußerste Schicht der Java EE Applikation an dieses Protokoll angepasst werden muss. Somit könnte hypothetisch ein Servlet so angepasst werden, dass es an diesem Port lauscht und über dieses Protokoll kommuniziert. Jedoch schränkt bereits die Servlet-Spezifikation Servlets auf HTTP ein.

> „All servlet containers must support HTTP as a protocol for requests and responses, but additional request/response-based protocols such as HTTPS (HTTP over SSL) may be supported.“[6, S.13]

Java EE verfolgt ein weiteres Prinzip, welches durch de JCA eingeführt wird: Resource Adapter. Ein Resource Adapter ist eine Java EE Komponente, die die Connector Architecture für ein spezifisches EIS implementiert. Die Integration wird hierbei in zwei Kategorien unterschieden [3]:

- Inbound Integration: externe Systeme initiieren Requests zu dem System,
- Outbound Integration: das System initiiert Requests zu anderen Systemen.

Abbildung 3 illustriert den JCA Ablauf beispielhaft.

Java Persistence API (JPA) Die JPA erlaubt es, Klassen zu erstellen, die Daten einer Datenbanktabelle abbilden – ein sogenanntes objektrelationales Mapping. Dabei können Entitätsklassen erstellt, modifiziert oder gelöscht werden. Um auf Daten zuzugreifen, kann die Java Persistence Query Language genutzt werden – eine SQL-ähnliche Sprache. Zusätzlich ist die Möglichkeit gegeben, entweder die Datenbanktabellen oder die Entitätsklassen automatisch zu generieren, ausgehend von dem Gegenpart.[11, S.26]

Exkurs MVC: In Java EE soll weitestgehend das Model View Control (MVC) Pattern angewandt werden. Bei diesem Pattern sollen Schnittstelle (View),

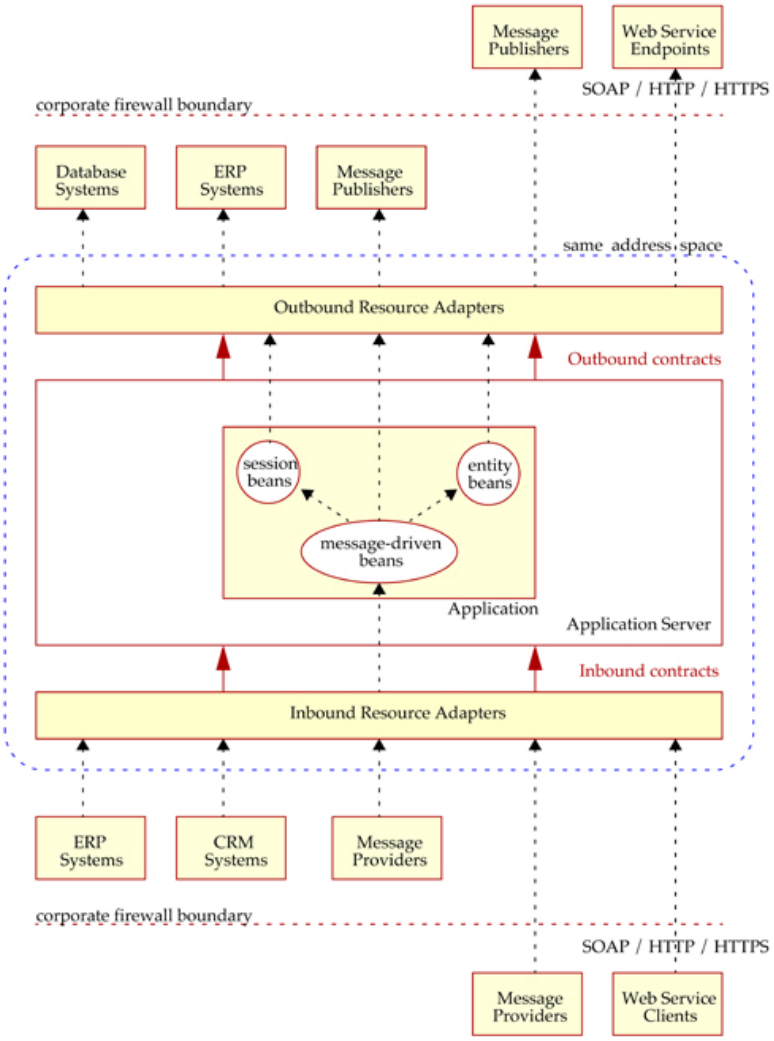

Abbildung 3: Ablauf der Java Connector Architecture[3, S.11-3]

Logik (Control) und Daten (Model) logisch voneinander getrennt sein. Auf Java EE abgebildet, gilt beispielsweise:

- View: Servlets / JSPs,
- Control: EJBs,
- Model: JPA.

4 Umsetzung

4.1 Machbarkeitsstudie

Dieses Kapitel stellt eine Art Machbarkeitsstudie dar, in der verschiedene hypothetische Lösungsvorschläge erarbeitet beziehungsweise so verändert werden, dass sie den Anforderungen genügen, um sich dann für eine zu entscheiden und diese konkret umzusetzen. Die folgenden Varianten basieren bereits aus den Anforderungen und gewonnenen Erkenntnissen. In allen Varianten soll Java EE eingesetzt werden, da potentielle Vorteile besonders effektiv genutzt werden könnten, beispielsweise:

- Erleichterte Erweiterbarkeit um z.B. ein Webfrontend,
- sicheres Threading,
- Multiuser Fähigkeiten,
- Security Fähigkeiten,
- Transaktionsmanagement (z.B. JPA).

4.1.1 Studie «RSS Aggregator»

Angenommen die Clients kommunizieren nicht direkt mit einem oder mehreren RSS Anbieter(n) sondern mit einer Middleware – einem RSS Aggregator respektive ein RSS Feed, der aus vorhandenen RSS Feeds neu komponiert wird. Dieser Aggregator soll in der Lage sein, nicht nur die konfigurierten RSS Feeds abzurufen und an die Clients auszugeben, sondern zusätzlich den Zeitpunkt der Ausgabe an die Clients zu vermerken. Bei jedem Abruf würden nur die RSS Nachrichten ausgegeben werden, die nach dem letzten Abruf (Zeitpunkt) erstellt wurden.

Damit würde das Problem vermieden, dass gleiche Nachrichten auf verschiedenen Clients unterschiedlich markiert werden (*gelesen / ungelesen*). Jede Nachricht ist nur auf jeweils einem Client gespeichert – auf dem, der sie abgerufen hat.

Umgesetzt in Java EE wäre als View (externe Schicht) ein Servlet denkbar, welches nach dem RSS Format aus den einzelnen RSS Nachrichten zusammengestellt wäre. Der dahinterliegende Controller würde eventuell in eine Datenbank über JPA den Zeitpunkt des letzten Abrufs durch einen Client speichern.

Tabelle 2: Beispielkommunikation über IMAP

Client	Server	Erklärung
(verbindet sich mit dem Server über einen Socket)		
	* OK IMAP4rev1	Begrüßung des Clients durch den Server
g001 login mrc secret		Anmeldung des Clients am Server
	g001 OK LOGIN completed	Bestätigung der Anmeldung durch den Server
g002 select inbox		Der Client wählt inbox als aktiven Ordner
	* 10 EXISTS * FLAGS (\Answered \Flagged \Deleted \Seen \Draft) g002 OK [READ-WRITE] SELECT completed	10 Mails vorhanden Ausgabe der Privilegien

Zusätzlich wäre der Aggregator auch Multiuser-fähig umsetzbar, indem der Aufruf beispielsweise parametrisiert werden würde
(z.B. http://domain/rss/aggregatedRSS?user=Paul)

Vorteilhaft zu bewerten, ist die passgenaue und triviale Umsetzung nach Java EE. Ein großer Nachteil, und deswegen nicht übernommen, stellt die disjunkte Verteilung der Nachrichten dar. Ideal wäre es, wenn sämtliche Clients alle Nachrichten abrufen können und sich untereinander synchronisieren.

4.1.2 Studie «IMAP für RSS»

IMAP (Internet Message Access Protokoll) stellt für den E-Mail-Verkehr ein Protokoll dar, mit welchem es ermöglicht wird, E-Mails zentral zu hosten und auch nach einer Offline-Bearbeitung zu synchronisieren. Der Ansatz dieser Studie bietet das gleiche Protokoll für RSS Nachrichten an, indem RSS Nachrichten ähnlich wie E-Mails benutzt werden.

> „The Internet Message Access Protocol, Version 4rev1 (IMAP4rev1) allows a client to access and manipulate electronic mail messages on a server. IMAP4rev1 permits manipulation of mailboxes (remote message folders) in a way that is functionally equivalent to local folders. IMAP4rev1 also provides the capability for an offline client to resynchronize with the server.“[2]

Tabelle 2 illustriert eine beispielhafte Client-Server-Kommunikation über IMAP.

Die Verwendung des bereits existenten IMAPs und Adaption für RSS impliziert mehrere Vorteile. Zum einen muss kein neues Protokoll entwickelt werden,

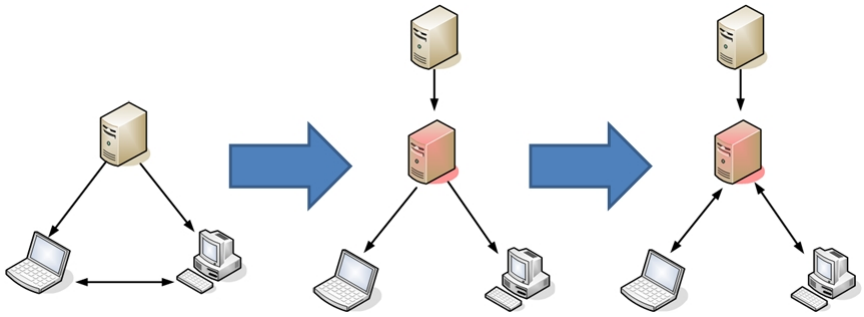

Abbildung 4: Vergleich der Studien

zum anderen auch kein Client an ein neues angepasst werden. Zudem ist IMAP de facto Standard, womit es zumindest in gängigen Clients implementiert und nutzbar ist.

Es lässt sich nun eine Middleware entwickeln, die sich zwischen Clients und RSS Anbietern positionieren lässt. Diese Middleware ruft konfigurierte RSS Anbieter ab und speichert dessen RSS Nachrichten. Auf der anderen Seite kann die Kommunikation zwischen Middleware und Client nun über IMAP stattfinden. Im Client wird ein E-Mail Konto konfiguriert, welches die Middleware adressiert. Dem E-Mail Client wird quasi „vorgegaukelt“, es handle sich um ein normales E-Mail-Postfach. Der Anwender kann nun RSS Nachrichten im Postfach als *gelesen* markieren oder löschen. Somit entsteht eine Art Adapter für RSS-Anbieter mit IMAP Funktionalität. Es existiert ein zentrales Feedback, welche RSS Nachrichten bereits gelesen wurden. Dieser Status kann nun mittels IMAP auf allen Clients synchronisiert werden.

Abbildung 4 illustriert anschaulich das ursprüngliche Konzept der RSS Nutzung mit merhreren Clients und vergleicht den Übergang bishin zur aktuellen Studie.

Der folgende Abschnitt beschreibt die technische Umsetzung dieser Studie. Entscheidend dabei ist ein weiteres Merkmal von IMAP.

> „The IMAP4rev1 protocol assumes a reliable data stream such as that provided by TCP. When TCP is used, an IMAP4rev1 server listens on port 143.“[2]

Dies erfordert bei der Java EE Umsetzung den Einsatz eines Resource Adapters.

4.2 Technische Vorraussetzungen

Folgende Vorraussetzungen müssen mindestens geschaffen werden, um das Projekt erfolgreich zu reproduzieren[1].

- Java EE 6 SDK Dafür ist die Voraussetzung eine JRE oder ein JDK 6.

[1] Respektive unter dieser Umgebung / diesen Versionen wurde die Applikation entwickelt und getestet.

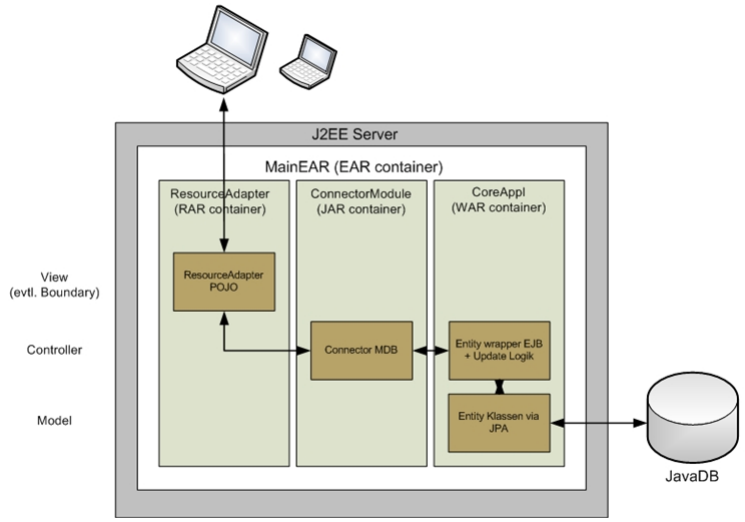

Abbildung 5: Containerbasierte Konzeption

- Als IDE kommt Netbeans zum Einsatz.
- Als Application Server wird Glassfish 3.0.1, welches in J2EE enthalten ist, verwendet. Dieser wird installiert und muss für dieses Projekt nicht weiter konfiguriert werden.

Anmerkung: Eine Installationsanleitung ist dem Anhang zu entnehmen.

4.3 Design und Implementierung

Die Praktische Umsetzung, beinhaltend Design und Implementierung, orientiert sich an folgenden Maßgaben:

1. Weitestgehende Wiederverwendung vorhandener und Wiederverwendbarkeit des neu erstellten Projekts oder einzelner Bestandteile (OOP Prinzip),
2. Verwendung von Design-Pattern (Best Practice) und
3. Umsetzung mittels des MVC Paradigmas (durch Java EE vorgegeben).

Nachfolgend wird der Aufbau und die Umsetzung containerweise erläutert. Abbildung 5 illustriert das Zusammenspiel und die Konzeption des Projekts per Container.

Tabelle 3 gibt eine Übersicht der bedeutendsten Ordner-Strukturen und Quell-Dateien des Projekts. (Sämtliche aufgelisteten Dateien wurden per Hand entwickelt und nicht generiert, es sei denn, es ist anders angegeben:
Anmerkung: Sämtliche Quelldateien sind auf der beigelegten DVD im Verzeichnis *source* enthalten. Ausgewählter Sourcecode ist zusätzlich dem Anhang zu entnehmen.

Tabelle 3: Auflistung der Ordnerstruktur und Quelldateien

Container / Pfad	Quelldatei
ConnectorModule	
src/conf	ejb-jar.xml
	sun-ejb-jar.xml
src/java/ejb	EntityWrapperLocal.java *(Kopie aus CoreAppl/src/java/ejb)*
src/java/imap	IMAPListenerBean.java
CoreAppl	
src/conf	persistence.xml
src/java/ejb	EntityWrapperBean.java
	EntityWrapperLocal.java
src/java/entity	Folder.java
	Foldertoitem.java
	Item.java
	Profile.java
	Repository.java
	Rssfeed.java
ResourceAdapter	
src/com/eis/tcp	*(Package wurde wiederverwendet. Die Änderungen werden in den nächsten Abschnitten beschrieben. Originalcode stammt von [12])*
	MessageWork.java
	ServerWork.java
	TCPActivationSpec.java
	TCPMessageListener.java
	TCPResourceAdapter.java
protocol/imap	IMAPElement.java
	IMAPElementChildIterator.java
	IMAPFolder.java
	IMAPFolderProxy.java
	IMAPMessage.java *(wiederverwendet. Originalcode von [14])*
	IMAPMessageProxy.java
	IMAPModel.java
	IMAPModelProxy.java
	IMAPProvider.java *(wiederverwendet. Originalcode von [14])*
	IMAPProxyConstants.java

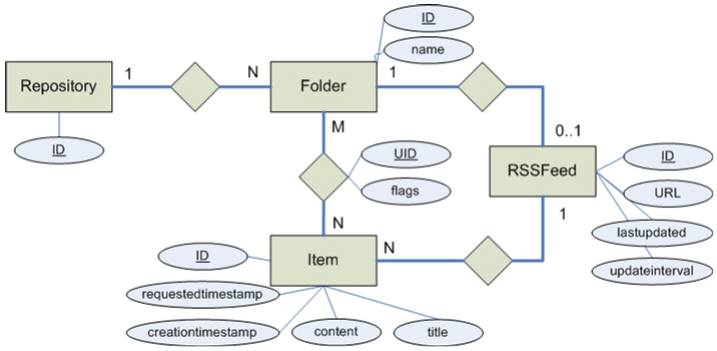

Abbildung 6: Datenmodell (Entity Relationship Modell)

4.3.1 MainEAR Container

Der Container *MainEAR* ist als EAR gepackt und repräsentiert das Gesamtprojekt (Hauptcontainer). Dieser wurde in Netbeans so konfiguriert, dass er alle drei folgenden Container beinhaltet. Somit muss nur dieser Container als eines zum Server deployed werden.

4.3.2 CoreAppl Container

Der *CoreAppl* Container ist als WAR gepackt. Es beinhaltet die Persistenz- und Business-Logik.

Persistenz Das Package *entity* beinhaltet alle Entity Beans. Diese werden automatisch bei jedem Deployment als JavaDB Tabellen mittels JPA abgebildet (generiert). Sollten die Datenbank Tabellen bereits bestehen, so ist in der *persistence.xml* konfiguriert, dass die Tabellen nicht überschrieben werden. Das Entity Relationship Modell (siehe Abbildung 6) illustriert, wie die Entity Beans in Beziehung stehen, sowie, welche Daten (Attribute) gekapselt werden. Die automatische Generierung von Datenbank Tabellen auf Basis der Entity Beans erfolgt im Glassfish der genutzten Version nur in einem WAR. Deshalb musste dieser zusätzliche Container im Projekt erstellt werden. Sonst hätte das Projekt auch aus lediglich zwei Container (zuzüglich des EAR) bestehen können.

Controller Die eigentliche Logik repräsentiert die Klasse *EntityWrapperBean*. Diese stellt Methoden bereit, um auf die Entitäten zuzugreifen, Daten zu modifizieren oder zu erstellen. Weiterhin können über die Methode *updateDataSources* sämtliche RSS Feeds, die in der Datenbank eingetragen sind, aktualisiert werden. Das heißt, dass die RSS Feeds abgerufen, neue Nachrichten gefiltert und persistiert werden. Dafür wird die RSS API benutzt[2]. Die Klasse implementiert das Interface *EntityWrapperLocal*. Dieses

[2]Diese ist zu finden unter [4] und liegt als als Package im Verzeichnis *CoreAppl/lib*.

wird bereitgestellt, um auch außerhalb des Containers auf ein Bean aus diesem Pool zuzugreifen.

4.3.3 ResourceAdapter Container

Der *ResourceAdapter* Container ist als RAR gepackt und implementiert die externe Schnittstelle respektive das IMAP. Dies funktioniert nach dem Prinzip der Inbound Integration. Für die Umsetzung des ResourceAdapter wurde ein vorhandenes Package [12] verwendet und adaptiert. Das Package beinhaltet einen TCP Resource Adapter, der auf den Port 143 konfiguriert wird. Bei der Ausführung wird ein Serversocket geöffnet und sobald eine Verbindung zu diesem Socket aufgenommen wird, setzt ein Threadhandling ein. Damit eine andere Klasse über eine potentielle Kommunikation benachrichtigt werden kann, können Klassen registriert werden, die das *TCPMessageListener* Interface implementieren. Im Wesentlichen verlangt dieses Interface eine *onMessage* Methode, die von allen regstrieren Klassen bei Eingang einer Kommunikation aufgerufen wird.
Das eigentliche Protokoll wird durch die Klasse *IMAPProvider* implementiert. Der Aufruf des *IMAPProvider* und der entsprechenden Methoden wurde in dem genannten Package adaptiert.
Für die Umsetzung der IMAP Funktionalität wird eine Marktevaluierung durchgeführt. Folgende IMAP Implementierungen lassen sich identifizieren:

JavaMail Ist im Wesentlichen auf die clientseitige Kommunikatioon beschränkt.[9]

Apache James stellt ein in Java implementierten Mail Server dar, der ab dem kommenden Release 3 eine IMAP Implementierung beinhalten wird. Diese Implementierung ist jedoch sehr „schwergewichtig“ in dem Sinne, dass komplett alle Bestandteile des IMAP Protokolls implementiert worden sind. Da aber nur ein Subset benötigt wird (es sollen RSS Nachrichten als gelesen markiert und gelöscht werden können) wird sich für eine andere Implementierung entschieden.[10]

rssImapServer-1.2 [14] Eine Applikation, die als Daemon designt ist. Diese Umsetzung liefert im Prinzip eine ähnliche Funktionalität wie die zu entwickelnde Applikation. Sie wird jedoch eher als Upgrade für ältere E-Mail Clients betrachtet, die keine eingebaute RSS Funktionalität mitbringen.

> „It is useful for adding RSS reading capability to older email clients that do support RSS natively (such as Outlook 2003) without resorting to plugins or mods that can affect the way the client operates.“[14]

Zusätzlich besitzt sie im Vergleich zur zu entwickelnden Applikation einige Restriktionen:

- RSS Nachrichten sind nicht speicherbar,

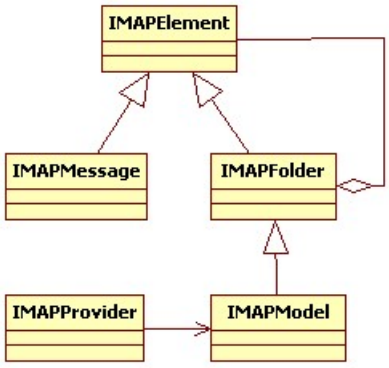

Abbildung 7: Design des IMAP Datenmodells mittels Composite Pattern

- RSS lassen sich nur lesen und ausgeben, jedoch den *gelesen* Status nicht persistieren. Es stellt praktisch ein Adapter dar.
- Die Applikation ist nicht in Java EE integriert, womit nicht die Vorteile generiert werden können, die im Abschnitt 4.1 aufgezählt und im Abschnitt 5.1 als potentielle Umsetzung erläutert werden.

Demenstprechend wird nur die IMAP Komponente des letzteren Packages wiederverwendet, welche in zwei Klassen (*IMAPMessage* und *IMAPProvider*) integriert und an einigen Stellen adaptiert wird. Diese Adaptionen beschränken sich auf den Aufruf des *IMAPModel*.
Dieser Provider benutzt ein *IMAPModel*, welches nach dem Composite Pattern einer Art Baumstruktur folgt, um eine Ordnerstruktur zu repräsentieren (vgl. Abbildung 7). Dieses Model existiert in zwei Varianten – als lokale und als Proxy Version. Die lokale Variante wurde in ersten Schritten experimentell manuell mit RSS Nachrichten und Verzeichnissen (*IMAPFolder*) „gefüllt“ und dem *IMAPProvider* übergeben.

Da sich die eigentliche Logik und Daten der Applikation aber in einem anderen Container (*CoreAppl*) befindet, wurde ein Proxy Model geschaffen. Der Provider greift auf dieses Proxy Model über die selbe Schnittstelle zu, wie bei der lokalen Variante. Somit ist dem Provider nicht bekannt, woher die Daten stammen. Dafür wird das Proxy Pattern angewandt (vgl. Abbildung 8).

Das *IMAPModelProxy* greift im Eigentlichen auf Methoden, der beim Resource Adapter registrierten MDB zu. Diese MDB wird im nächsten Abschnitt erläutert.

Um über den Inhalt eines *IMAPModel* iterieren zu können, werden verschiedene Iteratoren nach dem Iterator Muster bereitgestellt.

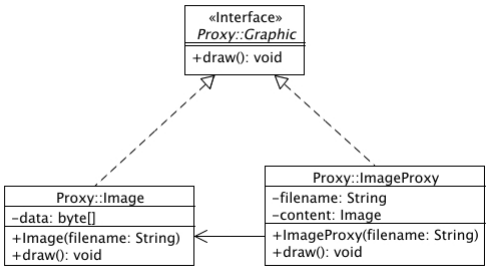

Abbildung 8: Beispielhafte Illustration des Proxy Pattern[13]

4.3.4 ConnectorModule Container

Der *ConnectorModule* Container (gepackt als JAR) bildet den Verbund zwischen Resource Adapter und Controller.

Damit die Klasse *IMAPListenerBean* von dem Resource Adapter über neue Nachrichten informiert wird, erbt die Klasse von *TCPMessageListener* und implementiert die Methode *onMessage*. Zusätzlich wird die erstellte MDB bei dem Resource Adapter über den Deployment Deskriptor angemeldet (*sun-ejb-jar.xml*; *ejb-jar.xml*). Der Resource Adapter ruft bei einer bestehenden IMAP Kommunikation die genannte *onMessage* Methode auf. Diese wiederum, um die jeweiligen Methoden des eigentlichen Controllers (*EntityWrapperBean* aus dem *CoreAppl* Container) aufzurufen, holt sich eine Instanz über einen sogenannten JNDI Lookup. Für diesen Zugriff muss dieser Container eine lokale Kopie des *EntityWrapperLocal* Interface beinhalten.

Die Kommunikation zwischen Resource Adapter (im speziellen dem *IMAP-ModelProxy* und den dazugehörigen Klassen) und dem *IMAPListenerBean* wird im *IMAPProxyConstants* Interface definiert. Das *IMAPModelProxy* sendet dem *IMAPListenerBean* über die *onMessage* Methode derartige Nachrichten. Das *IMAPListenerBean* ruft dann, nachdem die Nachricht über reguläre Ausdrücke (aus dem *IMAPProxyConstants* Interface) identifiziert wurde, die jeweiligen Methoden des Controllers (*EntityWrapperBean*) auf.

4.3.5 Ergebnisse

Die umgesetzte Hauptfunktionalität beinhaltet:

- Der Server aktualisiert abonnierte RSS Feeds selbständig,
- der Status einer RSS Message ist änderbar (z.B.. „*gelesen*“) via IMAP (z.B. in Outlook),
- das Synchronisieren zwischen Client und Server.

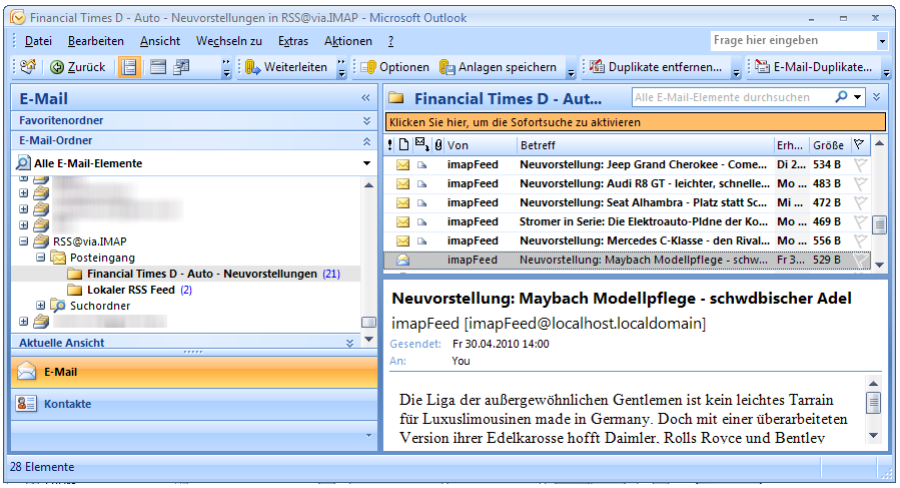

Abbildung 9: Outlook mit konfiguriertem (RSS) IMAP Account

Abbildung 9 illustriert die reale Nutzung der aktuellen Umsetzung. Abbildung 10 demonstriert einen hypothetischen sequentiellen Ablauf einer solchen IMAP Verbindung.

5 Endbetrachtung

5.1 Ausblick

Performanz: Bei der bisherigen Entwicklung wurde noch wenig Wert auf die Performanz gelegt. Das äußert sich momentan noch stark bei einer Kommunikation.

Konfigurierbarkeit: Nach aktuellem Stand muss beispielsweise der abzurufende RSS Feed noch manuell in die Datenbank eingetragen werden. Potentielle Lösungen wären die Einführung von Konfigurationsmails. Ein Nutzer könnte dem IMAP Server eine Nachricht senden, die den zu abonnierenden RSS Feed beschreibt. Eine weitere Möglichkeit bestünde in der Bereitstellung eines Online Interfaces.

Nutzen der Java EE Vorteile: Beispielsweise ermöglicht ein zusätzliches Online Inferface das Lesen unabhängig vom Client. Auf einer solchen Website könnten ebenso Statistiken (denkbar mittels Tag Clouds) über gelesene RSS Nachrichten dargestellt werden. Zudem ließe sich auch problemlos die diskutierte erste Variante als zusätzliches Servlet im CoreAppl Container integrieren. Die Abbildung 11 illustriert potentielle Änderungen am Container-Design.

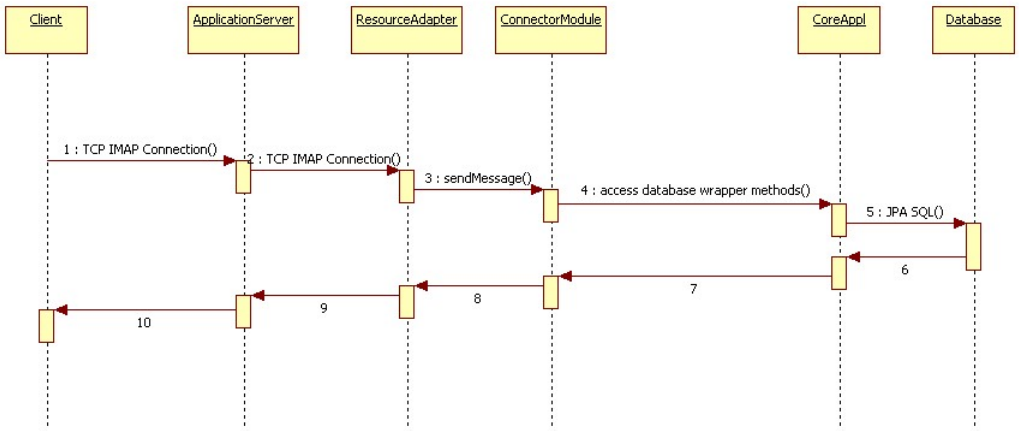

Abbildung 10: Sequenzdiagramm einer Beispielausführung

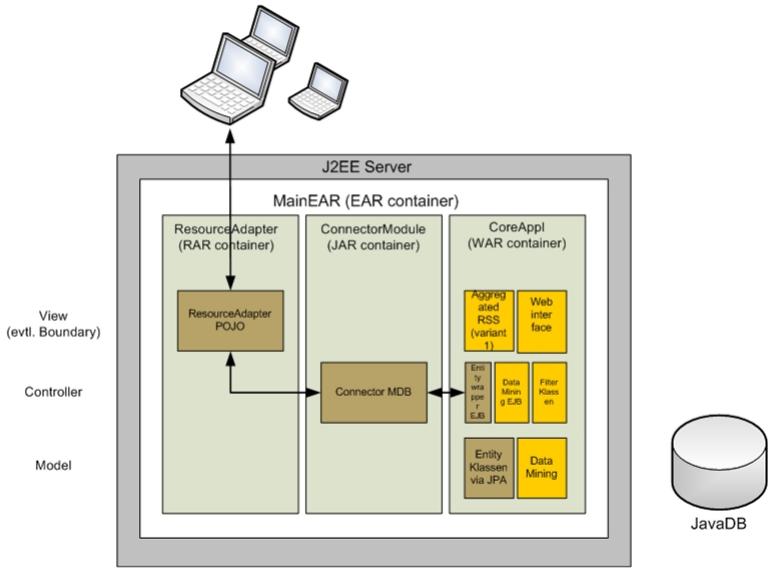

Abbildung 11: Angepasste Containerbasierte Konzeption (Ausblick)

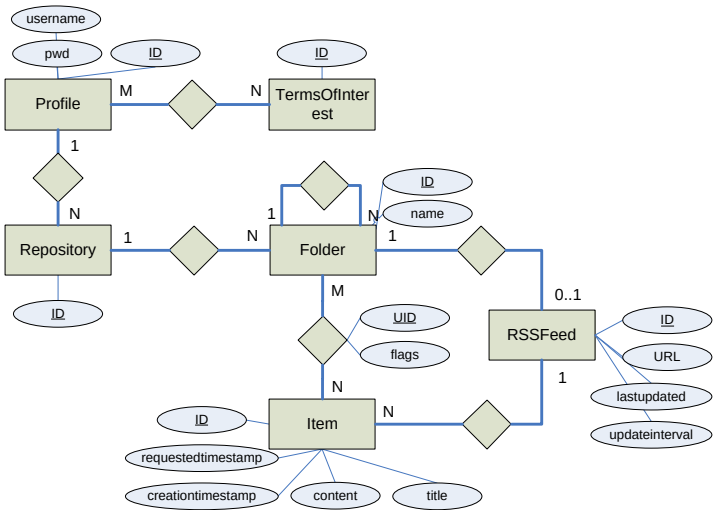

Abbildung 12: Angepasstes Entity Relationship Modell (Ausblick)

Lösen von anderen Problemen der RSS Nutzung: Wer beispielsweise verschiedene RSS Feeds mit ähnlichem Inhalt abonniert (z.B. Politik), dem wird auffallen, dass sich Nachrichten teilweise inhaltlich stark ähneln. Zudem erscheinen manchmal binnen weniger Minuten korrigierte Nachrichten (z.B. aufgrund von Rechtschreibfehlern) des selben Anbieters. Diese Nachrichten können gefiltert werden [3].

Data Mining: Das Feedback des Nutzers lässt weitere Betrachtungen zu. Somit kann eine Interessenbestimmung[4] genutzt werden, um zukünftige potentiell interessante RSS Nachrichten besser zu präsentieren[5].

Multiuser Die Applikation lässt sich problemlos für mehrere Benutzer adaptieren. Ein angepasstes ER Modell illustriert Abbildung 12.

[3] Eine potentielle Filterung überprüft die Ähnlichkeit von Betreff und/oder Inhalt der RSS-Nachricht (z.B. optional >90%) und den zeitlichen Abstand (z.B. optional innerhalb 5 Minuten).

[4] Angenommen jedem potentiellen Wort eines Betreffs werden zwei Zähler zugeordnet. Der erste wird hochgezählt, wenn das Wort in einer gelesenen RSS Nachricht enthalten ist respektive der zweite, wenn eine solche Nachricht ungelesen bleibt. Um eventuelle Füllwörter nicht als Interesse zu deklarieren, könnte das Verhältnis beider Zähler betrachtet werden. Würde ein Wort stark (z.B. optinal > 100) *gelesen* und zugleich *ungelesen* markiert, ist davon auszugehen, dass es sich nicht um ein Interesse handelt.

[5] Potentiell interessantere Nachrichten können in der Ausgabe an vorderer Stelle sortiert, mit einem Flag als «besonders wichtig» markiert werden oder der Betreff wird adaptiert (z.B. optional „[90% interessant] ...Betreff...“).

5.2 Fazit

Es wurde eine Möglichkeit geschaffen RSS Nachrichten über verschiedene Clients synchron zu halten. Die gesetzten Prinzipien (OOP Wiederverwendbarkeit) und Paradigmen sowie Anforderungen konnten umgesetzt werden. Zudem ist die Entwicklung so weit vorbereitet, dass potentielle Erweiterungen (des Ausblicks) in einem direkten nächsten Schritt umgesetzt werden können.

6 Anhang

6.1 Installation und Konfiguration

Um die Applikation nutzen zu können, müssen die technischen Vorraussetzungen installiert und konfiguriert werden (vgl. Abschnitt 4.2). Dazu gehört auch die Konfiguration einer Datenbank. Es genügt beispielsweise in Netbeans eine JavaDB Datenbank zu erstellen. Diese muss im Glassfish oder unter Netbeans als Persistence Unit mit der Bezeichnung „*CoreApplPU*“ eingerichtet werden.

Das Projekt wird deployed (*MainEAR*). Dies kann beispielsweise durch ein Import des Projekts *MainEAR* in Netbeans und dem Aufruf des jeweiligen Eintrags im Kontextmenü durchgeführt werden. Vorrausgesetzt keine störenden Faktoren wie beispielsweise eine Firewall oder ein anderes Tool, welches auf dem selben Port lauscht, existieren, sollte jetzt eine Verbindung (beispielsweise über Telnet, Outlook etc.) zu der Applikation möglich sein. RSS müssen nach aktuellem Stand manuell konfiguriert werden, indem die Tabellen *REPOSITORY* und *RSSFEED* mit den jeweiligen Einträgen gefüllt werden. Bei der Einrichtung eines E-Mail Clients ist die Angabe des Benutzers und des Passworts unrelevant, bei der Angabe des Ports wird der standard IMAP Port 143 verwendet.

6.2 Ausgewählter Quellcode

6.2.1 Quellcode des CoreAppl Containers

Listing 2: CoreAppl: persistence.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence
       " xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://java.sun.com/xml/ns/persistence␣http://
       java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
3    <persistence-unit name="CoreApplPU" transaction-type="JTA">
4      <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
5      <jta-data-source>coredatasource</jta-data-source>
6      <properties>
7        <property name="eclipselink.logging.level" value="FINEST"/>
8        <property name="eclipselink.ddl-generation" value="create-tables"/
             >
9      </properties>
10   </persistence-unit>
11 </persistence>
```

Listing 3: CoreAppl: EntityWrapperBean.java

```
1  package ejb;
2
3  import com.sun.cnpi.rss.elements.Rss;
4  import com.sun.cnpi.rss.parser.RssParser;
5  import com.sun.cnpi.rss.parser.RssParserException;
6
7  import com.sun.cnpi.rss.parser.RssParserFactory;
8  import entity.Folder;
9  import entity.Foldertoitem;
10 import entity.Item;
11 import entity.Repository;
12
13 import java.io.IOException;
```

```
14  import java.net.URL;
15  import java.text.DateFormat;
16  import java.text.ParseException;
17  import java.text.SimpleDateFormat;
18  import java.util.Collection;
19  import java.util.Date;
20  import java.util.Iterator;
21  import java.util.Locale;
22  import java.util.logging.Level;
23  import java.util.logging.Logger;
24  import javax.annotation.PostConstruct;
25  import javax.annotation.PreDestroy;
26  import javax.ejb.Stateless;
27  import javax.persistence.EntityManager;
28  import javax.persistence.EntityManagerFactory;
29  import javax.persistence.PersistenceUnit;
30  import javax.persistence.Query;
31
32  @Stateless
33  public class EntityWrapperBean implements EntityWrapperLocal {
34
35      @PersistenceUnit
36      private EntityManagerFactory emf;
37      private EntityManager em = null;
38
39      @PostConstruct
40      public void init() {
41          em = emf.createEntityManager();
42      }
43
44      @PreDestroy
45      public void destroy() {
46          if (em != null) {
47              em.close();
48          }
49      }
50      private static Logger logger =
51              Logger.getLogger(EntityWrapperBean.class.getName());
52
53      public Collection<Long> getFolderIDsByRepositoryID(long repositoryID
            ) {
54          Query query = em.createNamedQuery("Folder.selectIDByRepository")
                ;
55          query.setParameter("repository", em.find(Repository.class,
                repositoryID));
56          return query.getResultList();
57      }
58
59      public String getFolderNameByID(long folderID) {
60          Folder folder = (Folder) em.find(Folder.class, folderID);
61          return folder.getName();
62      }
63
64      public Collection<Long> getFolderChildrenByID(long folderID) {
65          Query query = em.createNamedQuery("Foldertoitem.selectIDByFolder
                ");
66          query.setParameter("folder", em.find(Folder.class, folderID));
67          return query.getResultList();
68      }
69
70      public String getItemSubject(long itemID) {
71          return em.find(Foldertoitem.class, itemID).getItem().getTitle();
72      }
73
74      public String getItemContent(long itemID) {
75          return em.find(Foldertoitem.class, itemID).getItem().getContent
                ();
76      }
```

```
77
78      public String getItemFlags(long itemID) {
79          return em.find(Foldertoitem.class, itemID).getFlags();
80      }
81
82      public void setItemFlags(long itemID, String newFlags) {
83          Foldertoitem foldertoitem = em.find(Foldertoitem.class, itemID);
84          foldertoitem.setFlags(newFlags);
85
86          em.merge(foldertoitem);
87          em.flush();
88      }
89
90      public Date getItemCreationdate(long itemID) {
91          return em.find(Foldertoitem.class, itemID).getItem().
                getCreationtimestamp();
92      }
93
94      public void updateDataSources() {
95          logger.info("EntityWrapperBean updateDataSources() invoked.");
96
97          assert emf != null;
98
99          try {
100             Repository r1 = em.find(Repository.class, 1L);
101
102             Iterator<Folder> folderIterator = r1.getFolderCollection().
                    iterator();
103
104             while (folderIterator.hasNext()) {
105                 Folder folder = folderIterator.next();
106                 System.out.println(folder.getName());
107                 System.out.println(folder.getRssfeed().getUrl());
108
109                 RssParser parser = null;
110                 Rss rss = null;
111
112                 try {
113                     parser = RssParserFactory.createDefault();
114                     rss = parser.parse(new URL(folder.getRssfeed().
                            getUrl()));
115                 } catch (IOException ex) {
116                     logger.log(Level.SEVERE, null, ex);
117                     continue;
118                 } catch (RssParserException ex) {
119                     logger.log(Level.SEVERE, null, ex);
120                     continue;
121                 }
122
123                 //Get all XML elements in the feed
124                 Collection items = rss.getChannel().getItems();
125                 if (items != null && !items.isEmpty()) {
126
127                     Iterator i = items.iterator();
128                     while (i.hasNext()) {
129                         com.sun.cnpi.rss.elements.Item item = (com.sun.
                                cnpi.rss.elements.Item) i.next();
130
131                         String title = item.getTitle().getText().trim();
132                         String content = item.getDescription().getText()
                                .trim();
133
134                         DateFormat formatter = new SimpleDateFormat("EEE
                                , dd MMM yyyy HH:mm:ss zzz", Locale.ENGLISH
                                );
135
136                         Date creationdate;
137                         try {
```

```
138                                 creationdate = formatter.parse(item.
                                        getPubDate().getText().trim());
139                             } catch (ParseException ex) {
140                                 logger.log(Level.SEVERE, null, ex);
141                                 continue;
142                             }
143
144                             Query query = em.createNamedQuery("Item.
                                    countByTitleAndCreationtimestamp");
145                             query.setParameter("creationtimestamp",
                                    creationdate);
146                             query.setParameter("title", title);
147                             if (Integer.valueOf(query.getSingleResult().
                                    toString()) == 0) {
148                                 Item dbitem = new Item();
149                                 dbitem.setTitle(title);
150                                 dbitem.setContent(content);
151                                 dbitem.setCreationtimestamp(creationdate);
152                                 dbitem.setRssfeed(folder.getRssfeed());
153                                 em.persist(dbitem);
154                                 em.flush();
155
156                                 // TODO: do this for every profile
                                        containing the respecting RSS feed
157                                 // TODO: insert already existing items on
                                        new creation of profile
158                                 Foldertoitem foldertoitem = new Foldertoitem
                                        ();
159                                 foldertoitem.setFolder(folder);
160                                 foldertoitem.setItem(dbitem);
161                                 foldertoitem.setFlags("");
162                                 em.persist(foldertoitem);
163                                 em.flush();
164                             }
165
166                         }
167                     }
168
169                 }
170
171             } catch (Exception ex) {
172                 logger.warning(ex.getLocalizedMessage());
173             }
174         }
175 }
```

Listing 4: CoreAppl: EntityWrapperLocal.java

```
1  package ejb;
2
3  import java.util.Collection;
4  import java.util.Date;
5
6  public interface EntityWrapperLocal {
7
8      void updateDataSources();
9
10     Collection<Long> getFolderIDsByRepositoryID(long repositoryID);
11
12     String getFolderNameByID(long folderID);
13
14     Collection<Long> getFolderChildrenByID(long folderID);
15
16     String getItemSubject(long itemID);
17
18     String getItemContent(long itemID);
19
20     String getItemFlags(long itemID);
```

```
21
22      void setItemFlags(long itemID, String newFlags);
23
24      Date getItemCreationdate(long itemID);
25  }
```

Listing 5: CoreAppl: Folder.java

```
1   package entity;
2
3   import java.io.Serializable;
4   import java.util.Collection;
5   import javax.persistence.Basic;
6   import javax.persistence.Column;
7   import javax.persistence.Entity;
8   import javax.persistence.GeneratedValue;
9   import javax.persistence.GenerationType;
10  import javax.persistence.Id;
11  import javax.persistence.JoinColumn;
12  import javax.persistence.ManyToOne;
13  import javax.persistence.NamedQueries;
14  import javax.persistence.NamedQuery;
15  import javax.persistence.OneToMany;
16  import javax.persistence.Table;
17
18  @Entity
19  @Table(name = "FOLDER")
20  @NamedQueries({
21      @NamedQuery(name = "Folder.findAll", query = "SELECT␣f␣FROM␣Folder␣f
            "),
22      @NamedQuery(name = "Folder.findById", query = "SELECT␣f␣FROM␣Folder␣
            f␣WHERE␣f.id␣=␣:id"),
23      @NamedQuery(name = "Folder.findByName", query = "SELECT␣f␣FROM␣
            Folder␣f␣WHERE␣f.name␣=␣:name"),
24      @NamedQuery(name = "Folder.selectIDByRepository", query = "SELECT␣f.
            id␣FROM␣Folder␣f␣WHERE␣f.repository␣=␣:repository")
25  })
26  public class Folder implements Serializable {
27      private static final long serialVersionUID = 1L;
28
29
30
31      @Id
32      @GeneratedValue(strategy=GenerationType.AUTO)
33      @Basic(optional = false)
34      @Column(name = "ID")
35      private Long id;
36      @Column(name = "NAME")
37      private String name;
38      @OneToMany(mappedBy = "folder")
39      private Collection<Foldertoitem> foldertoitemCollection;
40      @JoinColumn(name = "RSSFEED_ID", referencedColumnName = "ID")
41      @ManyToOne
42      private Rssfeed rssfeed;
43      @JoinColumn(name = "REPOSITORY_ID", referencedColumnName = "ID")
44      @ManyToOne
45      private Repository repository;
46
47      public Folder() {
48      }
49
50      public Folder(Long id) {
51          this.id = id;
52      }
53
54      public Long getId() {
55          return id;
56      }
```

```
57
58      public void setId(Long id) {
59          this.id = id;
60      }
61
62      public String getName() {
63          return name;
64      }
65
66      public void setName(String name) {
67          this.name = name;
68      }
69
70      public Collection<Foldertoitem> getFoldertoitemCollection() {
71          return foldertoitemCollection;
72      }
73
74      public void setFoldertoitemCollection(Collection<Foldertoitem>
            foldertoitemCollection) {
75          this.foldertoitemCollection = foldertoitemCollection;
76      }
77
78      public Rssfeed getRssfeed() {
79          return rssfeed;
80      }
81
82      public void setRssfeed(Rssfeed rssfeed) {
83          this.rssfeed = rssfeed;
84      }
85
86      public Repository getRepository() {
87          return repository;
88      }
89
90      public void setRepository(Repository repository) {
91          this.repository = repository;
92      }
93
94      @Override
95      public int hashCode() {
96          int hash = 0;
97          hash += (id != null ? id.hashCode() : 0);
98          return hash;
99      }
100
101     @Override
102     public boolean equals(Object object) {
103         // TODO: Warning - this method won't work in the case the id
                fields are not set
104         if (!(object instanceof Folder)) {
105             return false;
106         }
107         Folder other = (Folder) object;
108         if ((this.id == null && other.id != null) || (this.id != null &&
                !this.id.equals(other.id))) {
109             return false;
110         }
111         return true;
112     }
113
114     @Override
115     public String toString() {
116         return "tmp.Folder[id=" + id + "]";
117     }
118
119 }
```

Listing 6: CoreAppl: Foldertoitem.java

```
1  package entity;
2
3  import java.io.Serializable;
4  import javax.persistence.Basic;
5  import javax.persistence.Column;
6  import javax.persistence.Entity;
7  import javax.persistence.GeneratedValue;
8  import javax.persistence.GenerationType;
9  import javax.persistence.Id;
10 import javax.persistence.JoinColumn;
11 import javax.persistence.ManyToOne;
12 import javax.persistence.NamedQueries;
13 import javax.persistence.NamedQuery;
14 import javax.persistence.Table;
15
16 @Entity
17 @Table(name = "FOLDERTOITEM")
18 @NamedQueries({
19     @NamedQuery(name = "Foldertoitem.findAll", query = "SELECT f FROM
           Foldertoitem f"),
20     @NamedQuery(name = "Foldertoitem.findById", query = "SELECT f FROM
           Foldertoitem f WHERE f.id = :id"),
21     @NamedQuery(name = "Foldertoitem.findByFlags", query = "SELECT f
           FROM Foldertoitem f WHERE f.flags = :flags"),
22     @NamedQuery(name = "Foldertoitem.selectIDByFolder", query = "SELECT
           f.id FROM Foldertoitem f WHERE f.folder = :folder")
23 })
24 public class Foldertoitem implements Serializable {
25     private static final long serialVersionUID = 1L;
26     @Id
27     @GeneratedValue(strategy=GenerationType.AUTO)
28     @Basic(optional = false)
29     @Column(name = "ID")
30     private Long id;
31     @Column(name = "FLAGS")
32     private String flags;
33     @JoinColumn(name = "ITEM_ID", referencedColumnName = "ID")
34     @ManyToOne
35     private Item item;
36     @JoinColumn(name = "FOLDER_ID", referencedColumnName = "ID")
37     @ManyToOne
38     private Folder folder;
39
40     public Foldertoitem() {
41     }
42
43     public Foldertoitem(Long id) {
44         this.id = id;
45     }
46
47     public Long getId() {
48         return id;
49     }
50
51     public void setId(Long id) {
52         this.id = id;
53     }
54
55     public String getFlags() {
56         return flags;
57     }
58
59     public void setFlags(String flags) {
60         this.flags = flags;
61     }
62
63     public Item getItem() {
```

```
64          return item;
65      }
66
67      public void setItem(Item item) {
68          this.item = item;
69      }
70
71      public Folder getFolder() {
72          return folder;
73      }
74
75      public void setFolder(Folder folder) {
76          this.folder = folder;
77      }
78
79      @Override
80      public int hashCode() {
81          int hash = 0;
82          hash += (id != null ? id.hashCode() : 0);
83          return hash;
84      }
85
86      @Override
87      public boolean equals(Object object) {
88          // TODO: Warning - this method won't work in the case the id
                fields are not set
89          if (!(object instanceof Foldertoitem)) {
90              return false;
91          }
92          Foldertoitem other = (Foldertoitem) object;
93          if ((this.id == null && other.id != null) || (this.id != null &&
                !this.id.equals(other.id))) {
94              return false;
95          }
96          return true;
97      }
98
99      @Override
100     public String toString() {
101         return "tmp.Foldertoitem[id=" + id + "]";
102     }
103
104 }
```

Listing 7: CoreAppl: Item.java

```
1  package entity;
2
3  import java.io.Serializable;
4  import java.util.Collection;
5  import java.util.Date;
6  import javax.persistence.Basic;
7  import javax.persistence.Column;
8  import javax.persistence.Entity;
9  import javax.persistence.GeneratedValue;
10 import javax.persistence.GenerationType;
11 import javax.persistence.Id;
12 import javax.persistence.JoinColumn;
13 import javax.persistence.Lob;
14 import javax.persistence.ManyToOne;
15 import javax.persistence.NamedQueries;
16 import javax.persistence.NamedQuery;
17 import javax.persistence.OneToMany;
18 import javax.persistence.Table;
19 import javax.persistence.Temporal;
20 import javax.persistence.TemporalType;
21
22 @Entity
```

```
23  @Table(name = "ITEM")
24  @NamedQueries({
25      @NamedQuery(name = "Item.findByTitleAndCreationtimestamp", query = "
            SELECT i FROM Item i WHERE i.title = :title AND i.
            creationtimestamp = :creationtimestamp"),
26      @NamedQuery(name = "Item.countByTitleAndCreationtimestamp", query =
            "SELECT COUNT(i) FROM Item i WHERE i.title = :title AND i.
            creationtimestamp = :creationtimestamp"),
27      @NamedQuery(name = "Item.findAll", query = "SELECT i FROM Item i"),
28      @NamedQuery(name = "Item.findById", query = "SELECT i FROM Item i
            WHERE i.id = :id"),
29      @NamedQuery(name = "Item.findByContent", query = "SELECT i FROM Item
            i WHERE i.content = :content"),
30      @NamedQuery(name = "Item.findByTitle", query = "SELECT i FROM Item i
            WHERE i.title = :title"),
31      @NamedQuery(name = "Item.findByRequestedtimestamp", query = "SELECT
            i FROM Item i WHERE i.requestedtimestamp = :requestedtimestamp"
            ),
32      @NamedQuery(name = "Item.findByCreationtimestamp", query = "SELECT i
            FROM Item i WHERE i.creationtimestamp = :creationtimestamp")})
33  public class Item implements Serializable {
34
35      private static final long serialVersionUID = 1L;
36      @Id
37      @GeneratedValue(strategy = GenerationType.AUTO)
38      @Basic(optional = false)
39      @Column(name = "ID")
40      private Long id;
41      @Lob
42      @Column(name = "CONTENT")
43      private String content;
44      @Column(name = "TITLE", columnDefinition = "VARCHAR(1000)")
45      private String title;
46      @Column(name = "REQUESTEDTIMESTAMP")
47      @Temporal(TemporalType.TIMESTAMP)
48      private Date requestedtimestamp;
49      @Column(name = "CREATIONTIMESTAMP")
50      @Temporal(TemporalType.TIMESTAMP)
51      private Date creationtimestamp;
52      @OneToMany(mappedBy = "item")
53      private Collection<Foldertoitem> foldertoitemCollection;
54      @JoinColumn(name = "RSSFEED_ID", referencedColumnName = "ID")
55      @ManyToOne
56      private Rssfeed rssfeed;
57
58      public Item() {
59      }
60
61      public Item(Long id) {
62          this.id = id;
63      }
64
65      public Long getId() {
66          return id;
67      }
68
69      public void setId(Long id) {
70          this.id = id;
71      }
72
73      public String getContent() {
74          return content;
75      }
76
77      public void setContent(String content) {
78          this.content = content;
79      }
80
```

```
81      public String getTitle() {
82          return title;
83      }
84
85      public void setTitle(String title) {
86          this.title = title;
87      }
88
89      public Date getRequestedtimestamp() {
90          return requestedtimestamp;
91      }
92
93      public void setRequestedtimestamp(Date requestedtimestamp) {
94          this.requestedtimestamp = requestedtimestamp;
95      }
96
97      public Date getCreationtimestamp() {
98          return creationtimestamp;
99      }
100
101     public void setCreationtimestamp(Date creationtimestamp) {
102         this.creationtimestamp = creationtimestamp;
103     }
104
105     public Collection<Foldertoitem> getFoldertoitemCollection() {
106         return foldertoitemCollection;
107     }
108
109     public void setFoldertoitemCollection(Collection<Foldertoitem>
            foldertoitemCollection) {
110         this.foldertoitemCollection = foldertoitemCollection;
111     }
112
113     public Rssfeed getRssfeed() {
114         return rssfeed;
115     }
116
117     public void setRssfeed(Rssfeed rssfeed) {
118         this.rssfeed = rssfeed;
119     }
120
121     @Override
122     public int hashCode() {
123         int hash = 0;
124         hash += (id != null ? id.hashCode() : 0);
125         return hash;
126     }
127
128     @Override
129     public boolean equals(Object object) {
130         // TODO: Warning - this method won't work in the case the id
                fields are not set
131         if (!(object instanceof Item)) {
132             return false;
133         }
134         Item other = (Item) object;
135         if ((this.id == null && other.id != null) || (this.id != null &&
                !this.id.equals(other.id))) {
136             return false;
137         }
138         return true;
139     }
140
141     @Override
142     public String toString() {
143         return "tmp.Item[id=" + id + "]";
144     }
145 }
```

Listing 8: CoreAppl: Profile.java

```
1  package entity;
2
3  import java.io.Serializable;
4  import javax.persistence.Basic;
5  import javax.persistence.Column;
6  import javax.persistence.Entity;
7  import javax.persistence.GeneratedValue;
8  import javax.persistence.GenerationType;
9  import javax.persistence.Id;
10 import javax.persistence.NamedQueries;
11 import javax.persistence.NamedQuery;
12 import javax.persistence.Table;
13
14 @Entity
15 @Table(name = "PROFILE")
16 @NamedQueries({
17     @NamedQuery(name = "Profile.findAll", query = "SELECT␣p␣FROM␣Profile
       ␣p"),
18     @NamedQuery(name = "Profile.findById", query = "SELECT␣p␣FROM␣
       Profile␣p␣WHERE␣p.id␣=␣:id")})
19 public class Profile implements Serializable {
20     private static final long serialVersionUID = 1L;
21     @Id
22     @GeneratedValue(strategy=GenerationType.AUTO)
23     @Basic(optional = false)
24     @Column(name = "ID")
25     private Long id;
26
27     public Profile() {
28     }
29
30     public Profile(Long id) {
31         this.id = id;
32     }
33
34     public Long getId() {
35         return id;
36     }
37
38     public void setId(Long id) {
39         this.id = id;
40     }
41
42     @Override
43     public int hashCode() {
44         int hash = 0;
45         hash += (id != null ? id.hashCode() : 0);
46         return hash;
47     }
48
49     @Override
50     public boolean equals(Object object) {
51         // TODO: Warning - this method won't work in the case the id
               fields are not set
52         if (!(object instanceof Profile)) {
53             return false;
54         }
55         Profile other = (Profile) object;
56         if ((this.id == null && other.id != null) || (this.id != null &&
               !this.id.equals(other.id))) {
57             return false;
58         }
59         return true;
60     }
61
62     @Override
63     public String toString() {
```

```
64          return "tmp.Profile[id=" + id + "]";
65      }
66
67  }
```

Listing 9: CoreAppl: Repository.java

```
1   package entity;
2
3   import java.io.Serializable;
4   import java.util.Collection;
5   import javax.persistence.Basic;
6   import javax.persistence.Column;
7   import javax.persistence.Entity;
8   import javax.persistence.GeneratedValue;
9   import javax.persistence.GenerationType;
10  import javax.persistence.Id;
11  import javax.persistence.NamedQueries;
12  import javax.persistence.NamedQuery;
13  import javax.persistence.OneToMany;
14  import javax.persistence.Table;
15
16  @Entity
17  @Table(name = "REPOSITORY")
18  @NamedQueries({
19      @NamedQuery(name = "Repository.findAll", query = "SELECT␣r␣FROM␣
            Repository␣r"),
20      @NamedQuery(name = "Repository.findById", query = "SELECT␣r␣FROM␣
            Repository␣r␣WHERE␣r.id␣=␣:id")})
21  public class Repository implements Serializable {
22
23      private static final long serialVersionUID = 1L;
24      @Id
25      @GeneratedValue(strategy = GenerationType.AUTO)
26      @Basic(optional = false)
27      @Column(name = "ID")
28      private Long id;
29      @OneToMany(mappedBy = "repository")
30      private Collection<Folder> folderCollection;
31
32      public Repository() {
33      }
34
35      public Repository(Long id) {
36          this.id = id;
37      }
38
39      public Long getId() {
40          return id;
41      }
42
43      public void setId(Long id) {
44          this.id = id;
45      }
46
47      public Collection<Folder> getFolderCollection() {
48          return folderCollection;
49      }
50
51      public void setFolderCollection(Collection<Folder> folderCollection)
            {
52          this.folderCollection = folderCollection;
53      }
54
55      @Override
56      public int hashCode() {
57          int hash = 0;
58          hash += (id != null ? id.hashCode() : 0);
```

```
59          return hash;
60      }
61
62      @Override
63      public boolean equals(Object object) {
64          // TODO: Warning - this method won't work in the case the id
                fields are not set
65          if (!(object instanceof Repository)) {
66              return false;
67          }
68          Repository other = (Repository) object;
69          if ((this.id == null && other.id != null) || (this.id != null &&
                !this.id.equals(other.id))) {
70              return false;
71          }
72          return true;
73      }
74
75      @Override
76      public String toString() {
77          return "tmp.Repository[id=" + id + "]";
78      }
79  }
```

Listing 10: CoreAppl: Rssfeed.java

```
1   package entity;
2
3   import java.io.Serializable;
4   import java.util.Collection;
5   import java.util.Date;
6   import javax.persistence.Basic;
7   import javax.persistence.Column;
8   import javax.persistence.Entity;
9   import javax.persistence.GeneratedValue;
10  import javax.persistence.GenerationType;
11  import javax.persistence.Id;
12  import javax.persistence.NamedQueries;
13  import javax.persistence.NamedQuery;
14  import javax.persistence.OneToMany;
15  import javax.persistence.Table;
16  import javax.persistence.Temporal;
17  import javax.persistence.TemporalType;
18
19  @Entity
20  @Table(name = "RSSFEED")
21  @NamedQueries({
22      @NamedQuery(name = "Rssfeed.findAll", query = "SELECT␣r␣FROM␣Rssfeed
            ␣r"),
23      @NamedQuery(name = "Rssfeed.findById", query = "SELECT␣r␣FROM␣
            Rssfeed␣r␣WHERE␣r.id␣=␣:id"),
24      @NamedQuery(name = "Rssfeed.findByLastupdate", query = "SELECT␣r␣
            FROM␣Rssfeed␣r␣WHERE␣r.lastupdate␣=␣:lastupdate"),
25      @NamedQuery(name = "Rssfeed.findByUpdateinterval", query = "SELECT␣r
            ␣FROM␣Rssfeed␣r␣WHERE␣r.updateinterval␣=␣:updateinterval"),
26      @NamedQuery(name = "Rssfeed.findByUrl", query = "SELECT␣r␣FROM␣
            Rssfeed␣r␣WHERE␣r.url␣=␣:url")})
27  public class Rssfeed implements Serializable {
28
29      private static final long serialVersionUID = 1L;
30      @Id
31      @GeneratedValue(strategy = GenerationType.AUTO)
32      @Basic(optional = false)
33      @Column(name = "ID")
34      private Long id;
35      @Column(name = "LASTUPDATE")
36      @Temporal(TemporalType.TIMESTAMP)
37      private Date lastupdate;
```

```
38      @Column(name = "UPDATEINTERVAL")
39      private Integer updateinterval;
40      @Column(name = "URL")
41      private String url;
42      @OneToMany(mappedBy = "rssfeed")
43      private Collection<Item> itemCollection;
44      @OneToMany(mappedBy = "rssfeed")
45      private Collection<Folder> folderCollection;
46
47      public Rssfeed() {
48      }
49
50      public Rssfeed(Long id) {
51          this.id = id;
52      }
53
54      public Long getId() {
55          return id;
56      }
57
58      public void setId(Long id) {
59          this.id = id;
60      }
61
62      public Date getLastupdate() {
63          return lastupdate;
64      }
65
66      public void setLastupdate(Date lastupdate) {
67          this.lastupdate = lastupdate;
68      }
69
70      public Integer getUpdateinterval() {
71          return updateinterval;
72      }
73
74      public void setUpdateinterval(Integer updateinterval) {
75          this.updateinterval = updateinterval;
76      }
77
78      public String getUrl() {
79          return url;
80      }
81
82      public void setUrl(String url) {
83          this.url = url;
84      }
85
86      public Collection<Item> getItemCollection() {
87          return itemCollection;
88      }
89
90      public void setItemCollection(Collection<Item> itemCollection) {
91          this.itemCollection = itemCollection;
92      }
93
94      public Collection<Folder> getFolderCollection() {
95          return folderCollection;
96      }
97
98      public void setFolderCollection(Collection<Folder> folderCollection)
            {
99          this.folderCollection = folderCollection;
100     }
101
102     @Override
103     public int hashCode() {
104         int hash = 0;
```

```
105             hash += (id != null ? id.hashCode() : 0);
106             return hash;
107         }
108 
109         @Override
110         public boolean equals(Object object) {
111             // TODO: Warning - this method won't work in the case the id
                    fields are not set
112             if (!(object instanceof Rssfeed)) {
113                 return false;
114             }
115             Rssfeed other = (Rssfeed) object;
116             if ((this.id == null && other.id != null) || (this.id != null &&
                    !this.id.equals(other.id))) {
117                 return false;
118             }
119             return true;
120         }
121 
122         @Override
123         public String toString() {
124             return "tmp.Rssfeed[id=" + id + "]";
125         }
126 }
```

6.2.2 Quellcode des ResourceAdapter Containers

Listing 11: ResourceAdapter: ra.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <connector xmlns="http://java.sun.com/xml/ns/j2ee"
3    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
5  ␣␣http://java.sun.com/xml/ns/j2ee/connector_1_5.xsd" version="1.5">
6 
7    <display-name>TCP Inbound Adapter</display-name>
8    <vendor-name>TCP Solutions, Inc.</vendor-name>
9    <eis-type>TCP Request Receiver</eis-type>
10   <resourceadapter-version>1.0</resourceadapter-version>
11 
12   <resourceadapter>
13 
14     <resourceadapter-class>
15       com.eis.tcp.TCPResourceAdapter
16     </resourceadapter-class>
17 
18     <!-- ResourceAdapter default configuration properties -->
19     <config-property>
20       <config-property-name>Port</config-property-name>
21       <config-property-type>java.lang.Integer</config-property-type>
22       <config-property-value>143</config-property-value>
23     </config-property>
24 
25     <!-- The inbound resource adapter description -->
26     <inbound-resourceadapter>
27       <messageadapter>
28         <messagelistener>
29           <messagelistener-type>
30             com.eis.tcp.TCPMessageListener
31           </messagelistener-type>
32           <activationspec>
33             <activationspec-class>
34               com.eis.tcp.TCPActivationSpec
35             </activationspec-class>
36             <required-config-property>
37               <config-property-name>
38                 Message
```

```
39                </config-property-name>
40              </required-config-property>
41            </activationspec>
42          </messagelistener>
43        </messageadapter>
44      </inbound-resourceadapter>
45
46    </resourceadapter>
47  </connector>
```

Listing 12: ResourceAdapter: IMAPElement.java

```
1  package protocol.imap;
2
3  public class IMAPElement {
4
5      public enum Type {
6
7          IMAPElement,
8          IMAPFolder,
9          IMAPMessage
10     }
11
12     public Type getType() {
13         return Type.IMAPElement;
14     }
15
16     public void remove(IMAPElement in) throws Exception {
17         throw new Exception("remove not implemented here.");
18     }
19
20     public IMAPElement getChild(int index) throws Exception {
21         throw new Exception("getChild not implemented here.");
22     }
23
24     public int countChildren() throws Exception {
25         throw new Exception("countChildren not implemented here.");
26     }
27     protected String name = null;
28
29     public String getName() {
30         return name;
31     }
32     protected int UID = -1;
33
34     public int getUID() {
35         return UID;
36     }
37
38     public void add(IMAPElement in) throws Exception {
39         throw new Exception("add not implemented here.");
40     }
41
42     public IMAPElement(String name) {
43         this.name = name;
44     }
45
46     public int getSeqNumber() throws Exception {
47         throw new Exception("getSeqNumber not implemented here.");
48     }
49 }
```

Listing 13: ResourceAdapter: IMAPElementChildIterator.java

```
1  package protocol.imap;
2
3  import java.util.Iterator;
```

```
4  import protocol.imap.IMAPElement.Type;
5
6  public class IMAPElementChildIterator implements Iterator {
7
8      IMAPFolder folder = null;
9      int index = -1;
10     final static int IMAPElement = 0;
11     final static int IMAPFolder = 1;
12     final static int IMAPMessage = 2;
13     Type childFilter = Type.IMAPElement;
14
15     public IMAPElementChildIterator(IMAPFolder in, Type childFilter) {
16         this(in);
17         this.childFilter = childFilter;
18     }
19
20     public IMAPElementChildIterator(IMAPFolder in) {
21         this.folder = in;
22         this.determineNextElement();
23     }
24
25     public boolean hasNext() {
26         return index != -1;
27     }
28
29     public IMAPElement next() {
30         if (this.index == -1) {
31             return null;
32         }
33
34         int lastIndex = this.index;
35         this.determineNextElement();
36
37         return folder.getChild(lastIndex);
38     }
39
40     private void determineNextElement() {
41         int lastChildIndex = folder.countChildren() - 1;
42
43         while (++this.index <= lastChildIndex) {
44             if ((childFilter == Type.IMAPElement || childFilter == Type.
                   IMAPFolder) && (folder.getChild(this.index).getType()
                   == Type.IMAPFolder)) {
45                 return;
46             } else if ((childFilter == Type.IMAPElement || childFilter
                   == Type.IMAPMessage) && (folder.getChild(this.index).
                   getType() == Type.IMAPMessage)) {
47                 return;
48             }
49         }
50
51         this.index = -1;
52     }
53
54     public void remove() {
55         // TODO
56     }
57 }
```

Listing 14: ResourceAdapter: IMAPFolder.java

```
1  package protocol.imap;
2
3  import java.util.Vector;
4
5  public class IMAPFolder extends IMAPElement {
6
7      public IMAPFolder(String name) {
```

```
8           super(name);
9       }
10
11      @Override
12      public Type getType() {
13          return IMAPElement.Type.IMAPFolder;
14      }
15      Vector v = new Vector();
16
17      @Override
18      public void add(IMAPElement in) {
19          v.add(in);
20      }
21
22      public void remove(IMAPElement in) {
23          v.remove(in);
24      }
25
26      public IMAPElement getChild(int index) {
27          return (IMAPElement) v.get(index);
28      }
29
30      public int countChildren() {
31          return v.size();
32      }
33  }
```

Listing 15: ResourceAdapter: IMAPFolderProxy.java

```
1   package protocol.imap;
2
3   import com.eis.tcp.TCPResourceAdapter;
4   import java.util.Vector;
5
6   public class IMAPFolderProxy extends IMAPFolder {
7
8       private long folderID;
9       TCPResourceAdapter ra = null;
10
11      public IMAPFolderProxy(Long folderID, TCPResourceAdapter ra) {
12          super("");
13          this.folderID = folderID;
14          this.ra = ra;
15      }
16
17      @Override
18      public String getName() {
19          return (String) ra.sendMessage(IMAPProxyConstants.FOLDER_GETNAME
                + IMAPProxyConstants.SEPARATOR + String.valueOf(folderID))
                ;
20      }
21      protected Vector<Long> childrenIDs = null;
22
23      private Vector<Long> getRemoteChildren() {
24          if (childrenIDs == null) {
25              childrenIDs = (Vector) ra.sendMessage(IMAPProxyConstants.
                    FOLDER_GETCHILDREN + IMAPProxyConstants.SEPARATOR +
                    String.valueOf(folderID));
26          }
27          return childrenIDs;
28      }
29
30      public IMAPElement getChild(int index) {
31          return new IMAPMessageProxy(getRemoteChildren().get(index).
                intValue(), ra, index + 1);
32      }
33
34      public int countChildren() {
```

```
35          return getRemoteChildren().size();
36      }
37  }
```

Listing 16: ResourceAdapter: IMAPMessageProxy.java

```
1   package protocol.imap;
2
3   import com.eis.tcp.TCPResourceAdapter;
4   import java.util.Date;
5
6   public class IMAPMessageProxy extends IMAPMessage implements
        IMAPProxyConstants {
7
8       TCPResourceAdapter ra = null;
9
10      public IMAPMessageProxy(int UID, TCPResourceAdapter ra, int
            seqNumber) {
11          super("", seqNumber, UID);
12          this.ra = ra;
13      }
14
15      private String remoteSubject = null;
16
17      @Override
18      public String getSubject() {
19          if (remoteSubject == null) {
20              remoteSubject = (String) ra.sendMessage(IMAPProxyConstants.
                    ITEM_GETSUBJECT + IMAPProxyConstants.SEPARATOR + String
                    .valueOf(getUID()));
21          }
22          return remoteSubject;
23      }
24
25      private String remoteText = null;
26
27      @Override
28      public String getText() {
29          if (remoteText == null) {
30              remoteText = (String) ra.sendMessage(IMAPProxyConstants.
                    ITEM_GETCONTENT + IMAPProxyConstants.SEPARATOR + String
                    .valueOf(getUID()));
31          }
32          return remoteText;
33      }
34
35      private Date remoteCreationdate = null;
36
37      @Override
38      public Date getCreationdate() {
39          if (remoteCreationdate == null) {
40              remoteCreationdate = (Date) ra.sendMessage(
                    IMAPProxyConstants.ITEM_GETCREATIONDATE +
                    IMAPProxyConstants.SEPARATOR + String.valueOf(getUID())
                    );
41          }
42          return remoteCreationdate;
43      }
44
45      private String remoteFlags = null;
46
47      @Override
48      public String getFlags() {
49          if (remoteFlags == null) {
50              remoteFlags = (String) ra.sendMessage(IMAPProxyConstants.
                    ITEM_GETFLAGS + IMAPProxyConstants.SEPARATOR + String.
                    valueOf(getUID()));
51          }
```

```
52          return remoteFlags;
53      }
54
55      @Override
56      public void addFlag(String flag) {
57          getFlags();
58          remoteFlags += flag;
59
60          ra.sendMessage(IMAPProxyConstants.ITEM_SETFLAGS +
                IMAPProxyConstants.SEPARATOR + String.valueOf(getUID()) +
                IMAPProxyConstants.SEPARATOR + remoteFlags);
61      }
62  }
```

Listing 17: ResourceAdapter: IMAPModel.java

```
1   package protocol.imap;
2
3   public class IMAPModel extends IMAPFolder {
4       public IMAPModel() {
5           super("INBOX");
6       }
7   }
```

Listing 18: ResourceAdapter: IMAPModelProxy.java

```
1   package protocol.imap;
2
3   import com.eis.tcp.TCPResourceAdapter;
4   import java.util.Vector;
5
6   public class IMAPModelProxy extends IMAPModel {
7
8       /**
9        * Die REPOSITORY_ID soll in spÃ¤teren Releases (bei EinfÃ¼hrung der
10       * Multi-User FÃ¤higkeit) variabel konfigurierbar werden.
11       */
12      final static String REPOSITORY_ID = "1";
13
14      TCPResourceAdapter ra = null;
15      protected Vector<Long> folderIDs = null;
16
17      public IMAPModelProxy(TCPResourceAdapter in) {
18          super();
19          this.ra = in;
20
21          ra.sendMessage(IMAPProxyConstants.MODEL_UPDATEDATASOURCES);
22          folderIDs = (Vector) ra.sendMessage(IMAPProxyConstants.
                MODEL_GETFOLDERIDS + IMAPProxyConstants.SEPARATOR +
                REPOSITORY_ID);
23      }
24
25      @Override
26      public IMAPElement getChild(int index) {
27          return new IMAPFolderProxy(folderIDs.get(index), ra);
28      }
29
30      @Override
31      public int countChildren() {
32          return folderIDs.size();
33      }
34  }
```

Listing 19: ResourceAdapter: IMAPProxyConstants.java

```
1   package protocol.imap;
```

```
2
3  public interface IMAPProxyConstants {
4
5      final static String PROLOG = "imap";
6
7      final static String SEPARATOR = "::";
8
9
10     final static String MODEL_UPDATEDATASOURCES
11             = "model" + SEPARATOR + "updatedatasources";
12
13     final static String MODEL_UPDATEDATASOURCES_REGEX
14             = PROLOG + SEPARATOR + MODEL_UPDATEDATASOURCES;
15
16
17     final static String MODEL_GETFOLDERIDS
18             = "model" + SEPARATOR + "getfolderids";
19
20     final static String MODEL_GETFOLDERIDS_REGEX
21             = PROLOG + SEPARATOR + MODEL_GETFOLDERIDS + SEPARATOR + "
                   ([0-9]+)";
22
23
24     final static String FOLDER_GETNAME = "folder" + SEPARATOR + "getname
           ";
25
26     final static String FOLDER_GETNAME_REGEX
27             = PROLOG + SEPARATOR + FOLDER_GETNAME + SEPARATOR + "
                   ([0-9]+)";
28
29
30     final static String FOLDER_GETCHILDREN
31             = "folder" + SEPARATOR + "getchildren";
32
33     final static String FOLDER_GETCHILDREN_REGEX
34             = PROLOG + SEPARATOR + FOLDER_GETCHILDREN + SEPARATOR + "
                   ([0-9]+)";
35
36
37     final static String ITEM_GETSUBJECT = "item" + SEPARATOR + "
           getsubject";
38
39     final static String ITEM_GETSUBJECT_REGEX
40             = PROLOG + SEPARATOR + ITEM_GETSUBJECT + SEPARATOR + "
                   ([0-9]+)";
41
42
43     final static String ITEM_GETCONTENT = "item" + SEPARATOR + "
           getcontent";
44
45     final static String ITEM_GETCONTENT_REGEX
46             = PROLOG + SEPARATOR + ITEM_GETCONTENT + SEPARATOR + "
                   ([0-9]+)";
47
48
49     final static String ITEM_GETCREATIONDATE
50             = "item" + SEPARATOR + "getcreationdate";
51
52     final static String ITEM_GETCREATIONDATE_REGEX
53             = PROLOG + SEPARATOR + ITEM_GETCREATIONDATE + SEPARATOR + "
                   ([0-9]+)";
54
55
56     final static String ITEM_GETFLAGS = "item" + SEPARATOR + "getflags";
57
58     final static String ITEM_GETFLAGS_REGEX
59             = PROLOG + SEPARATOR + ITEM_GETFLAGS + SEPARATOR + "([0-9]+)
                   ";
```

```
60
61
62      final static String ITEM_SETFLAGS = "item" + SEPARATOR + "setflags";
63
64      final static String ITEM_SETFLAGS_REGEX
65              = PROLOG + SEPARATOR + ITEM_SETFLAGS + SEPARATOR + "([0-9]+)
                    "
66              + SEPARATOR + "(.+)";
67  }
```

6.2.3 Quellcode des ConnectorModule Containers

Listing 20: ConnectorModule: ejb-jar.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ejb-jar version="2.1" xmlns="http://java.sun.com/xml/ns/j2ee"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
5  ␣␣http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd">
6    <display-name>Test MDB</display-name>
7    <enterprise-beans>
8      <!-- a message driven descriptor -->
9      <message-driven>
10       <display-name>IMAPListenerBean</display-name>
11       <ejb-name>IMAPListenerBean</ejb-name>
12       <ejb-class>imap.IMAPListenerBean</ejb-class>
13       <!-- The message listener interface -->
14       <messaging-type>com.eis.tcp.TCPMessageListener</messaging-type>
15       <transaction-type>Container</transaction-type>
16       <!-- the values for the Activation Spec JavaBean -->
17       <activation-config>
18         <activation-config-property>
19           <activation-config-property-name>
20             Message
21           </activation-config-property-name>
22           <activation-config-property-value>
23             imap::
24           </activation-config-property-value>
25         </activation-config-property>
26       </activation-config>
27     </message-driven>
28
29   </enterprise-beans>
30   <assembly-descriptor>
31     <container-transaction>
32       <method>
33         <ejb-name>IMAPListenerBean</ejb-name>
34         <method-name>*</method-name>
35       </method>
36       <trans-attribute>NotSupported</trans-attribute>
37     </container-transaction>
38   </assembly-descriptor>
39 </ejb-jar>
```

Listing 21: ConnectorModule: sun-ejb-jar.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE sun-ejb-jar PUBLIC "-//Sun␣Microsystems,␣Inc.//DTD␣Application
       ␣Server␣9.0␣EJB␣3.0//EN" "http://www.sun.com/software/appserver/
       dtds/sun-ejb-jar_3_0-0.dtd">
3  <sun-ejb-jar>
4    <enterprise-beans>
5      <ejb>
6        <ejb-name>IMAPListenerBean</ejb-name>
7        <mdb-resource-adapter>
8          <resource-adapter-mid>MainEAR#ResourceAdapter</resource-adapter-
               mid>
```

```
9         </mdb-resource-adapter>
10      </ejb>
11    </enterprise-beans>
12  </sun-ejb-jar>
```

Listing 22: ConnectorModule: IMAPListenerBean.java

```
1   package imap;
2
3   import javax.ejb.CreateException;
4   import javax.ejb.EJBException;
5   import javax.ejb.MessageDrivenBean;
6   import javax.ejb.MessageDrivenContext;
7
8   import com.eis.tcp.TCPMessageListener;
9   import ejb.EntityWrapperLocal;
10  import java.util.logging.Level;
11  import java.util.logging.Logger;
12
13  import java.util.regex.Matcher;
14  import java.util.regex.Pattern;
15  import javax.naming.InitialContext;
16  import protocol.imap.IMAPProvider;
17  import protocol.imap.IMAPProxyConstants;
18
19  /**
20   * Ein Endpoint fÃ¼r den TCP Resource Adapter.
21   */
22  public class IMAPListenerBean implements MessageDrivenBean,
        TCPMessageListener, IMAPProxyConstants {
23
24      private MessageDrivenContext context = null;
25
26      private EntityWrapperLocal entitywrapper;
27
28      private static Logger logger =
29              Logger.getLogger(IMAPProvider.class.getName());
30
31      public void ejbCreate() throws CreateException, EJBException {
32      }
33
34      public void ejbRemove() throws EJBException {
35      }
36
37      public void setMessageDrivenContext(final MessageDrivenContext
            context)
38              throws EJBException {
39          this.context = context;
40      }
41
42      public Object onMessage(String message) {
43          try {
44              logger.info("IMAPListenerBean.onMessage");
45              InitialContext initialcontext = new InitialContext();
46              entitywrapper = (EntityWrapperLocal) initialcontext.lookup("
                    java:global/MainEAR/CoreAppl/EntityWrapperBean!ejb.
                    EntityWrapperLocal");
47
48              Pattern pattern = null;
49              Matcher matcher = null;
50
51              pattern = Pattern.compile(IMAPProxyConstants.
                    MODEL_UPDATEDATASOURCES_REGEX);
52              matcher = pattern.matcher(message);
53              if (matcher.matches()) {
54                  logger.info(IMAPProxyConstants.MODEL_UPDATEDATASOURCES);
55                  entitywrapper.updateDataSources();
56                  return null;
```

```
57                  }
58
59                  pattern = Pattern.compile(IMAPProxyConstants.
                        MODEL_GETFOLDERIDS_REGEX);
60                  matcher = pattern.matcher(message);
61                  if (matcher.matches()) {
62                      logger.info(IMAPProxyConstants.MODEL_GETFOLDERIDS);
63                      return entitywrapper.getFolderIDsByRepositoryID(Long.
                            valueOf(matcher.group(1)));
64                  }
65
66                  pattern = Pattern.compile(IMAPProxyConstants.
                        FOLDER_GETNAME_REGEX);
67                  matcher = pattern.matcher(message);
68                  if (matcher.matches()) {
69                      logger.info(IMAPProxyConstants.FOLDER_GETNAME);
70                      return entitywrapper.getFolderNameByID(Long.valueOf(
                            matcher.group(1)));
71                  }
72
73                  pattern = Pattern.compile(IMAPProxyConstants.
                        FOLDER_GETCHILDREN_REGEX);
74                  matcher = pattern.matcher(message);
75                  if (matcher.matches()) {
76                      logger.info(IMAPProxyConstants.FOLDER_GETCHILDREN);
77                      return entitywrapper.getFolderChildrenByID(Long.valueOf(
                            matcher.group(1)));
78                  }
79
80                  pattern = Pattern.compile(IMAPProxyConstants.
                        ITEM_GETSUBJECT_REGEX);
81                  matcher = pattern.matcher(message);
82                  if (matcher.matches()) {
83                      logger.info(IMAPProxyConstants.ITEM_GETSUBJECT);
84                      return entitywrapper.getItemSubject(Long.valueOf(matcher
                            .group(1)));
85                  }
86
87                  pattern = Pattern.compile(IMAPProxyConstants.
                        ITEM_GETCONTENT_REGEX);
88                  matcher = pattern.matcher(message);
89                  if (matcher.matches()) {
90                      logger.info(IMAPProxyConstants.ITEM_GETCONTENT);
91                      return entitywrapper.getItemContent(Long.valueOf(matcher
                            .group(1)));
92                  }
93
94                  pattern = Pattern.compile(IMAPProxyConstants.
                        ITEM_GETFLAGS_REGEX);
95                  matcher = pattern.matcher(message);
96                  if (matcher.matches()) {
97                      logger.info(IMAPProxyConstants.ITEM_GETFLAGS);
98                      return entitywrapper.getItemFlags(Long.valueOf(matcher.
                            group(1)));
99                  }
100
101                 pattern = Pattern.compile(IMAPProxyConstants.
                        ITEM_GETCREATIONDATE_REGEX);
102                 matcher = pattern.matcher(message);
103                 if (matcher.matches()) {
104                     logger.info(IMAPProxyConstants.ITEM_GETCREATIONDATE);
105                     return entitywrapper.getItemCreationdate(Long.valueOf(
                            matcher.group(1)));
106                 }
107
108                 pattern = Pattern.compile(IMAPProxyConstants.
                        ITEM_SETFLAGS_REGEX);
109                 matcher = pattern.matcher(message);
```

```
110                 if (matcher.matches()) {
111                     logger.info(IMAPProxyConstants.ITEM_SETFLAGS);
112                     entitywrapper.setItemFlags(Long.valueOf(matcher.group(1)
                            ), matcher.group(2).trim());
113                     return null;
114                 }
115             } catch (Exception e) {
116                 logger.severe("Controller lookup failed.");
117                 logger.severe(e.getLocalizedMessage());
118             }
119 
120             return null;
121         }
122     }
```

Referenzen

[1] Easy2Sync für Outlook. http://www.easy2sync.de/de/produkte/e2s4o.php [zugegriffen: 18.08.2010].

[2] INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1 [zugegriffen: 19.08.2010]. http://tools.ietf.org/html/rfc3501, March 2003.

[3] J2EE™ Connector Architecture Specification Version 1.5 [zugegriffen: 18.08.2010]. http://java.sun.com/j2ee/connector/, November 2003.

[4] RSS Utilities: A Tutorial. http://java.sun.com/developer/technicalArticles/javaserverpages/rss_utilities/ [zugegriffen: 19.08.2010], August 2003.

[5] The J2EE™ 1.4 Tutorial for NetBeans™ IDE 4.1. http://netbeans.org/files/documents/4/442/J2EE-NBTutorial.pdf [zugegriffen: 20.08.2010], Mai 2005.

[6] Java™ Servlet Specification Version 2.5 MR6. http://jcp.org/aboutJava/communityprocess/mrel/jsr154/index2.html [zugegriffen: 18.08.2010], August 2007.

[7] rss2imap. http://directory.fsf.org/project/rss2imap/ [zugegriffen: 18.08.2010], Oktober 2007.

[8] Bloglines. http://www.bloglines.com/ [zugegriffen: 18.08.2010], 2008.

[9] Javamail. http://www.oracle.com/technetwork/java/index-jsp-139225.html [zugegriffen: 20.08.2010], 2009.

[10] James project. http://james.apache.org/ [zugegriffen: 20.08.2010], Juni 2010.

[11] Your First Cup: An Introduction to the Java EE Platform. download.oracle.com/javaee/6/firstcup/doc/firstcup.pdf [zugegriffen: 18.08.2010], Juni 2010.

[12] D. C. Dürr. Inbound Resource Adapter. http://www.prozesse-und-systeme.de/jcaInbound.html [zugegriffen: 20.08.2010].

[13] E. F. . E. Freeman. *Entwurfsmuster von Kopf bis Fuß*, volume 1. O'Reilly, 2005.

[14] R. Johnson. RSS->IMAP Server. http://rssimapserver.sourceforge.net/ [zugegriffen: 18.08.2010], 2006.

[15] V. Patryshev. Java Package Processes All RSS Formats. http://www.devx.com/Java/Article/21415 [zugegriffen: 18.08.2010], Juni 2004.