

Feedfilter Developer Guide

Lars Trieloff
Hasso-Plattner-Institute
lars@trieloff.net

ABSTRACT

The Feedfilter developer guide explains how to install and extend the Goshaky Feedfilter. Feedfilter is an application that uses modern Information Filtering and Retrieval Technology to categorize RSS and Atom feeds from websites according to custom, user-defined profiles and creates RSS feeds as output.

The basic underlying Information Filtering and Retrieval Models of Feedfilter are the enhanced Topic-based Vector Space Model and the K-Nearest-Neighbors-classification heuristics.

1. INTRODUCTION

With the advent of cheap mass-storage for personal computers, computers for the first time could store more information than humans could manage without technical help. The popularization of the Internet and the World Wide Web worsened this problem, because an ever increasing amount of information, both relevant and irrelevant was available.

Information Retrieval technology was seen as a partial solution to the problem, but not the only one. Syndication Feeds popularized through formats like RSS and Atom allow users to get information from interesting websites without having to search for, but as users subscribe to more and more feeds, the amount of new information becomes unmanageable again.

Information Filtering technology can help users separating relevant information from irrelevant information by evaluating new documents against an existing profile.

The Goshaky Feedfilter application is an example of leveraging Information Filtering technology by bringing together syndication feeds as a source of information, user-defined categories as measurement for relevancy and Information Filtering technology as a means to determining the correct classification for new information on the fly. More than that, its modular component-oriented architecture allows it to act as a framework for testing Information Filtering and Retrieval models in real-world applications.

In the remainder of this guide, Section 2, "Getting Started" will show how to install Goshaky Feedfilter. Afterwards Section 3, "Developing Feedfilter" aims at developers who want to dive further into Feedfilter and learn how to obtain the latest source code snapshot and how to compile Feedfilter from the sources. In Section 4, "System Architecture" the main system architecture is described from a conceptual point of view. This is required for understanding how the concepts explained in Section 5, "Implementation Concepts" fit in. To put the pieces together, read Section 6, "Module and Dependency Guide" which explains what modules implement which functionality and what dependencies were used when implementing Feedfilter. The the final part of the paper, Section 7, "Outlook" we show future prospects and related works.

2. GETTING STARTED

Currently Feedfilter is a completely self-contained application, there is no need for external databases or other data sources. This also marks its biggest limitation up to now, because there is no persistence layer either and everything is computed in main memory.

In order to set up Feedfilter, following prerequisites must be fulfilled:

- A Java Virtual Machine, version 1.4 or higher, available from Sun Microsystems [3].
- A Servlet container conforming to version 2.3 of the Java Servlet Specification, available the Apache Software Foundation [7].
- The Feedfilter Web Application Archive (WAR), available from the Goshaky Maven Repository [5].

After downloading and installing the Java Virtual Machine and the Servlet Container, you can simply deploy the WAR-file by copying it to the webapps-directory of the Servlet engine.

In case of a default installation of Apache Tomcat, you can access the application under the URL `http://localhost:8080/feedfilter-1.0-SNAPSHOT`.

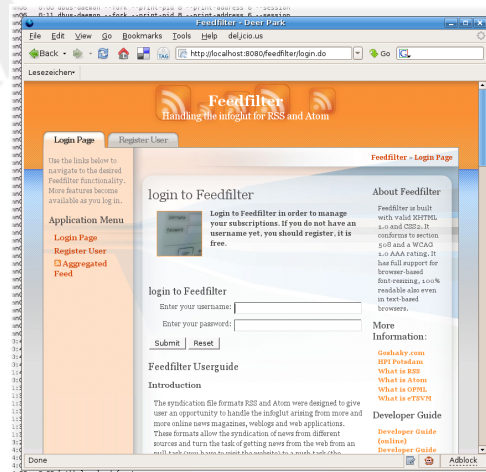


Figure 1. The Installed Feedfilter Application

Note

Please keep aware of the fact that Feedfilter currently does not offer a persistence layer and all changes made to the data will be lost after restarting the application container.

3. DEVELOPING FEEDFILTER

Goshaky Feedfilter is developed as an Open-Source application and the source code of Feedfilter and the Information Filtering, Information Retrieval and Document classification library is available for any user.

In order to get started with the development process, you need following prerequisites:

- A Java Software Development Kit, version 1.4 or higher, available from Sun Microsystems [4].
- Apache Maven 2, an advanced build tool, available from the Apache Software Foundation [6].
- Subversion, a Version Control System, available from tigris.org [22].

Setting up the development environment includes installing the prerequisites and following steps:

1. Check out the Goshaky Classifier library sources. This library is used for all Information Filtering and Retrieval tasks as well as document classification.

In the following code-listings and console screen captures, the backslash character is used to break up long command line input for better readability. So a backslash character (/) followed by a line break should be typed as a single space.

```
# svn co \
  http://goshaky.com/svn/goshaky/classifier
```

2. Check out the Feedfilter sources.

```
# svn co \
  http://goshaky.com/svn/feedfilter/trunk \
  feedfilter
```

3. Compile, test and install the classifier library sources. Maven will take care of the correct build process, downloading dependencies and will build all sub-projects as well.

```
# cd classifier
# mvn install
```

4. Compile, test and install the Feedfilter application.

```
# cd ../feedfilter
# mvn install
```

5. Let Maven generate IDE-specific settings for you.

```
# mvn eclipse:eclipse #for the Eclipse IDE
# mvn idea:idea #for IntelliJ IDEA
```

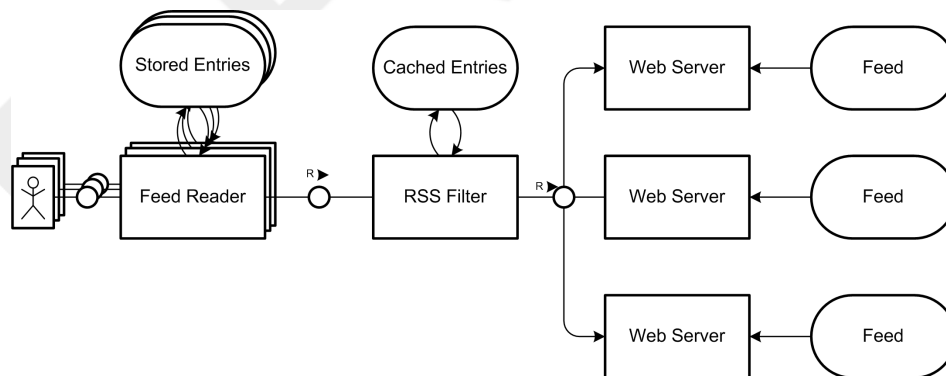
After having set up the development environment and developing extensions to the application or the Goshaky Classifier library, you are invited to generate patches and send them to the author. In order to generate a patch, you have to issue following command from the project's working directory:

```
# svn patch > your-patch-name.patch
```

4. SYSTEM ARCHITECTURE

The Feedfilter application and the underlying libraries are designed with a maximum of extensibility, flexibility and modularity in mind. It should be possible to exchange one component by another component, for example to allow for different persistence strategies, web application frameworks or conceptual models.

In fact, one goal was the possibility to test different Information Filtering and Retrieval models as well as classification schemes without having to change crucial parts of the application.



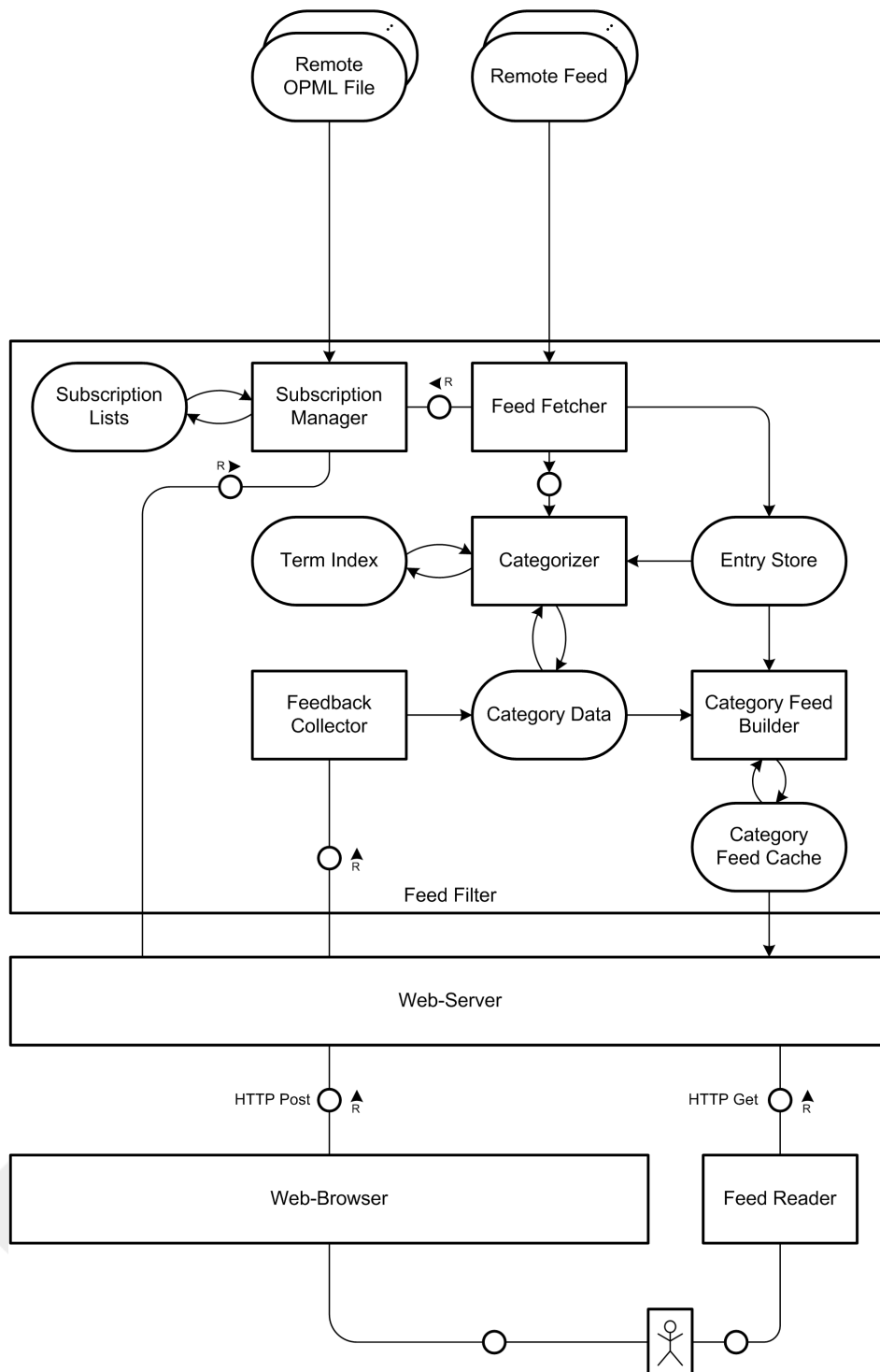
Notation: Fundamental Modeling Concepts Block Diagram

Figure 2. High-level architecture

The block diagram above shows the general architecture of the system at runtime. A number of web server serves different feeds from different web sites. These are read by the RSS Filter application, which will cache and categorize them locally. Users are now able to read these cached and categorized entries

using their feed readers which may be desktop or web-based applications.

The feed-reader application as such is front-end-agnostic, there is only a web-interface for managing subscriptions and categorization. The detailed architectural description below illustrates this fact.



Notation: Fundamental Modeling Concepts Block Diagram

Figure 3. Filter Architecture

To start from the end user, the user uses his web browser to access the subscription manager through a web interface. This subscription manager keeps a list of feeds¹ and feed groups² monitored for an user. A continuously run component, the feed fetcher will get all subscriptions from all users and start downloading new entries. These entries are stored in an

entry store for later reference and the categorizer component is invoked.

This categorizer or classifier uses the stored entry data and a self-maintained term index, which keeps document representations specific to the selected Information Filtering and Retrieval model in order to find the correct category for an entry.

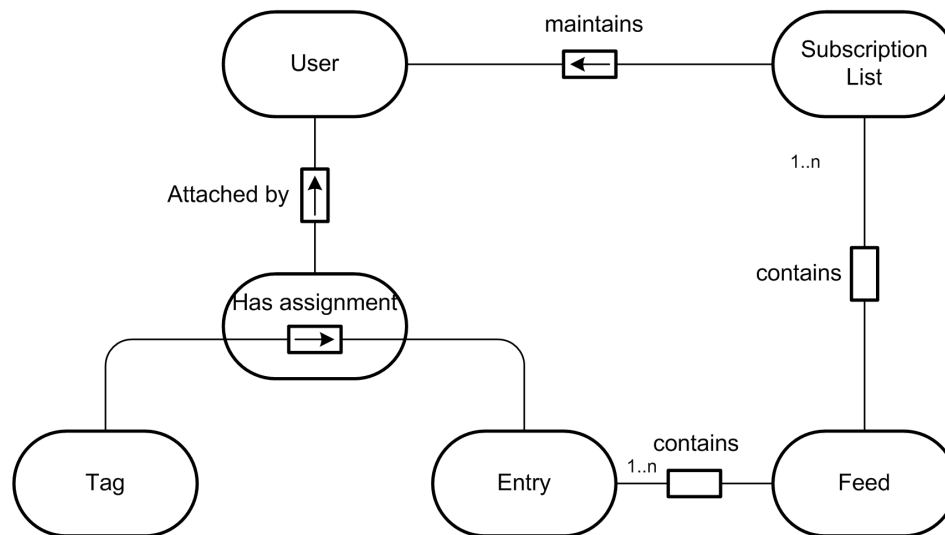
¹RSS and Atom feeds
²OPML files

The categorizer accesses for this purpose the category database which is maintained by the categorizer by automatically categorizing feeds and the feedback collector, who categorizes on the users behalf.

The category data is also accessed by the category feed builder, a component that will create new feeds from the saved entries specific for users and categories. In order to minimize server load, a feed cache will store generated feeds for a configurable amount of time.

In the end, the user is able to subscribe to the feeds generated by the category feed builder. When reading the generated feeds in his feed reader he will find a list of categories this entry belongs to at the end of the entry's content as well as a link to the feedback collector's web interface that allows the addition or removal of categories.

The category data model in this application used is the tagging-model. Categories are mutually independent "tags", and an entry can have any number of tags available. This concept was popularized by web sites like del.icio.us, which offers a bookmark service based on tags.



Notation: Fundamental Modeling Concepts Entity-Relationship-Diagram

Figure 4. Data Model

An entry has a number of tags assigned, but every tag assignment is associated with an user. For the later categorization of entries a category-facade is built up that emulates for every user and tag an own category.

5. IMPLEMENTATION CONCEPTS

The Goshaky classifier library used for categorizing new entries automatically is build upon two important concepts, the enhanced Topic-based Vector Space Model and the k-nearest neighbors classification heuristics.

The enhanced Topic-based Vector Space Model

The enhanced Topic-based Vector Space Model is a document representation model suitable for Information Filtering and Retrieval that represents documents as vectors in a vector space. This model has been described in detail by Dominik Kuroпка in [1]. One basic form of the enhanced Topic-based Vector Space Model is the classic Vector Space Model, which is described in [2].

The similarity of documents is measured as the cosine of the angle between two document vectors. This easy-to-calculate measurement returns document similarities between 0 (orthogonal vectors) and 1 (documents are regarded identical).

In the classic Vector Space Model each dimension in the vector space represents one term. The enhanced Topic-based Vector Space Model takes into consideration that one term can have multiple interpretations and that these interpretations share relations and interdependencies.

Therefore the topic is the basic building unit of this model and one dimension in the vector space represents one topic. Each interpretation vector consists of the relevant topics, which can be generated from a topic map. A document is a collection of interpretations, which are added to form a document vector.

Using powerful topic-maps, linguistic phenomena like morphology, synonymy, homography, metonymy, hyponymy and meronymy can be detected and represented.

The k-Nearest Neighbors Classification Heuristics

Classifying a document given a profile of other documents and their classification is a common task in Information Filtering, because it allows users to build up an easy-to-understand profile by classifying documents they know and letting the system use this classification database for its own decisions.

The k-nearest neighbors classification heuristics works as follows:

1. Get a new document
2. Calculate the similarity of this document compared to all existing documents in the profile
3. Select the k (which is a configurable variable) most similar documents. These are called neighbors.
4. Find out what category most of the neighbors belong to.
5. The category most of the neighbors belong to is assigned to the classified document. In the variant used by Goshaky Feedfilter, a limited number of categories is selected and applied to the document.

6. MODULE AND DEPENDENCY GUIDE

This section should be seen as a starter to understanding the different modules involved in building Feedfilter and their internal and external dependencies.

Goshaky Classifier Modules

classifier-core

The basic API for the classifier. This contains interfaces like Document, Category, Similarity and a default implementation of a document repository. The classifier API is application- and implementation independent. You should be able to implement any document representation model and any classification algorithm using the concepts described in this API. Furthermore the API is not tied to any application, you can use it for applications like Feedfilter, but other Information Filtering or Information Retrieval systems can use this API as well.

etsvm-core

Basic API for the enhanced Topic-based Vector Space Model and a default, in-memory implementation of the Model. The API defines concepts like Topic, Interpretation and the ETSVMDocument³. The default implementation of the InterpretationProvider does not take any existing topic maps into consideration.

k-nearest-neighbors-classifier

Interface definition and basic implementation of a k-nearest-neighbors classification algorithm. This implementation will work with any document representation conforming to the classifier-core API.

The Feedfilter application build upon those modules and consists of following modules.

feedfilter-model

Definition of the basic data model of the application, consisting of users, subscriptions, tags, feeds and entries. This module contains only the interfaces of the data model, or a conceptual description. Currently there is only one implementation of feedfilter-model, which is provided by feedfilter-core, but other implementations are imaginable.

feedfilter-core

Basic implementation of the data model and implementation of an in-memory document repository. This module conforms to the Subscription Lists, Subscription Manager and Entry Store in the Filter Architecture diagram. The classes in this module implement the interfaces defined in feedfilter-model. In the future there may be another implementation of the feedfilter-model interfaces, perhaps one using relational databases as persistence layer.

feedfilter-tagger

Implementation of the tagging system of the application. This module conforms to the Category Data and parts of the Feedback Collector in the Filter Architecture diagram.

feedfilter-fetcher

This module conforms to the Feed Fetcher in the Filter Architecture diagram and implements a task queue that will download new entries and invoke the classifier, implemented by the Goshaky classifier library.

feedfilter-web

The web-application serving as front end to all components. In particular the Category Feed Builder is implemented by this module.

The application uses a list of external libraries to provide functionality of the system. The most important external dependencies are shown in the following list.

External Dependencies

plexus

A lightweight application server and inversion-of-control container. This container allows weak coupling of modules and configurable system composition by modifying simple XML configuration files. Plexus [8] was chosen over other dependency-injection containers like Spring [9], Picocontainer [11] or Nanocontainer [10] for its flexibility and the large number of modules like plexus-taskqueue that make building applications upon Plexus easy.

rome

This library is used in order to download, parse and create RSS and Atom feeds. It is divided into a core library and a fetcher library. Thanks to the weak coupling it is possible to replace this library with another feed parsing library like Informa [12]. ROME [13] was chosen in this implementation, because it is more widely used than Informa and there is the possibility to create new feeds using ROME.

struts

The web-application framework used in this application, which provides a MVC-architecture for the web-application and some additional features like easy internationalization. Thanks to the weak coupling it is possible to replace this component with any other web-application framework like JSF [14] or Webwork [15]. Struts [16] was mainly chosen, because it is the most widely used

³for consistency among the module names, which are all lowercase and among class names, which are camel case, the acronym used for the enhanced Topic-based Vector Space Model does sometimes not conform to the standard spelling eTSVM.

web-application framework for J2EE and the primary author of Feedfilter has already used Strus before.

oscache

Provides a cache implementation that works as Servlet filter and will cache all response for one hour. This implements the Category Feed Cache in the Filter Architecture Diagram. OSCache [17] was selected, because it was the most easy to find Servlet filter to provide caching maintained in a professional project.

nekohtml

A HTML parser that is used in removing formatting from the input documents. The main reason for selecting NekoHTML [18] over other HTML parsers like JTidy [19] was the option to parse HTML documents using the SAX processing model, which is ideal for extracting textual contents from a document.

The following diagram shows the dependencies between the internal modules and their external requirements. White boxes denote components of the Feedfilter application, light-gray boxes are for modules of the Goshaky classifier library and dark-gray boxes are for external dependencies.

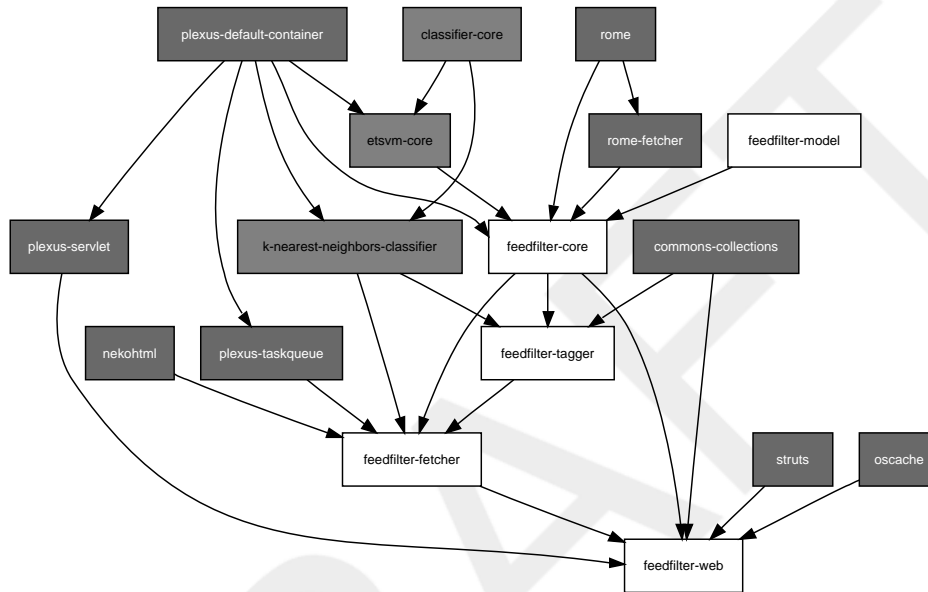


Figure 5. Dependency overview

7. OUTLOOK

The Goshaky Feedfilter application shows the viability of a feed-filtering service that uses advanced Information Filtering and Retrieval Technology.

Its modular design allows the implementation of other Information Filtering and Retrieval concepts as well as different front ends. The separation of the Goshaky classifier library makes it possible to use this library in other applications.

An application that is in many respects similar, is Agro, the Aggregator [21] by Nick Lothian, which is based on the Information Filtering and Retrieval Library Classifier4J [20], both by Nick Lothian. Agro is essentially a single-user system and uses probability-based classification. Opposed to Feedfilter, it offers an online interface to read aggregated entries, but no tagging interface.

Further development should concentrate on two tasks: Providing an implementation of the enhanced Topic-based Vector Space Model that works in external memory, i.e. in relational Database in order to provide persistence and reduce memory consumption, which is currently the most important limitation of the system. The second task is the implementation of other Information Filtering and Retrieval concepts as well

as classification schemes in order to allow an evaluation of the performance of these applications in a real-world-environment.

Further research could use the existing library and application as a testbed framework for demonstrating and evaluating those models.

REFERENCES

1. *Modelle zur Repräsentation natürlichsprachlicher Dokumente*. Dominik Kuroпка. Logos Verlag. Berlin . 2004.
2. *Modern Information Retrieval*. Ricardo Baeza-Yates and Berthier Ribeiro-Neto. Pearson Education. Singapore. 1999.
3. 01-18-2005. Sun Microsystems. *Download Java Software* [<http://java.com/en/download/index.jsp>].
4. 01-18-2005. Sun Microsystems. *Download Java 2 Platform Standard Edition 5.0* [<http://java.sun.com/j2se/1.5.0/download.jsp>].
5. 01-18-2005. Lars Trelloff. *Goshaky Maven Repository* [<http://www.goshaky.com/m2/>].
6. 01-18-2005. Apache Software Foundation. *Download Maven 2.0.2* [<http://maven.apache.org/download.html>].

7. 01-18-2005. Apache Software Foundation. *Apache Tomcat 5 Downloads*
[<http://tomcat.apache.org/download-55.cgi>].
8. 01-18-2005. The Codehaus. *Plexus - Home*
[<http://plexus.codehaus.org/>].
9. 01-18-2005. www.springframework.org. *Spring Application Framework*
[<http://www.springframework.org/>].
10. 01-18-2005. The Codehaus. *NanoContainer - Home*
[<http://nanocontainer.codehaus.org/>].
11. 01-18-2005. The Codehaus. *PicoContainer - Home*
[<http://www.picocontainer.org/>].
12. 01-18-2005. Niko Schmuck. *Informa: RSS Library for Java - Overview*
[<http://informa.sourceforge.net/>].
13. 01-18-2005. Sun Microsystems. *rome: All feeds lead to Rome* [<https://rome.dev.java.net/>].
14. 01-18-2005. Sun Microsystems. *JavaServer Faces Technology*
[<http://java.sun.com/j2ee/javaserverfaces/>].
15. 01-18-2005. OpenSymphony. *WebWork - WebWork*
[<http://www.opensymphony.com/webwork/>].
16. 01-18-2005. Apache Software Foundation. *Welcome - Apache Struts Project* [<http://struts.apache.org/>].
17. 01-18-2005. OpenSymphony. *OSCache - OSCache*
[<http://www.opensymphony.com/oscache/>].
18. 01-18-2005. Andy Clark. *NekoHTML*
[<http://people.apache.org/~andyc/neko/doc/html/>].
19. 01-18-2005. Fabrizio Giustina. *JTidy - JTidy*
[<http://jtidy.sourceforge.net/>].
20. 01-18-2005. Nick Lothian. *Classifier4J - Classifier4J*
[<http://classifier4j.sourceforge.net/>].
21. 01-18-2005. Nick Lothian. *Agro the Aggregator*
[<http://www.mackmo.com/agro/index.jsp>].
22. 01-18-2005. tigris.org. *subversion: Subversion Packages*
[http://subversion.tigris.org/project_packages.html].