

January 19, 2009



RssOwl

<http://www.rssowl.org>

Team Members

Braunhofer Matthias

Moling Omar

Strumpflohner Juri

Software Process Management

1 Introduction

RSSOwl is an RSS and Atom News reader / aggregator that is able to collect and display feeds from different compliant websites. Besides the basic functionality to collect and display feeds it has the following functionalities:

- Advanced search functionality that allows to find news feeds that satisfy a certain search query
- Organize news feeds in categories
- Auto-update of news feeds
- Links of news items can directly be sent to friends
- Internal browser that allows to read news items
- Labeling of news items with keywords
- Import / export of feeds
- ...

RSSOwl is written in Java as an Eclipse RCP application. It is licensed under the Eclipse Public License¹ and consists of 67,514 lines of code (LOC), not considering comment and blank lines. Some further technical features are the following:

- DB4O² object database for storing data persistently
- Apache HttpClient for connecting to the Internet
- JDom for dealing with XML data
- Apache Lucene for searching capabilities

2 Mission

The main mission for the project could be defined as follows:

Addition, modification and deletion of features of an Open Source project with at least 50,000 LOC.

For this purpose the main features implemented during the development have been the following:

Added features:

- Statistics view: shows statistics about news feeds (i.e. popularity, read percentage,...)
- News item rating

Modified feature:

- Filtering of news items according to their assigned labels

Deleted feature:

- Internal browser: instead, the default external browsers is used

The specific implementation of these core features will be outlined in the subsequent sections.

1 <http://www.opensource.org/licenses/eclipse-1.0.php>

2 <http://www.db4o.com/>

3 Development process

The project has been developed by using agile software methods, in particular by taking into consideration some XP practices. The applied practices during the development process are the following:

- Planning game
With the beginning of each week the user stories to be implemented have been selected from the list of available ones, depending on the priority and estimated effort.
- Short releases
The development cycle has been structured to have weekly iterations (sprints), always having a running system, which would have allowed to immediately stop the development (if the customer would have decided to do so).
- Simple design
Design has been kept as simple as possible.
- Refactoring
Refactoring has been taken place after the implementation of the user stories in order to keep the design simple and maintainable.
- Test-driven-development (TDD)
Tests have been written before the actual implementation.
- Collective code ownership
The source code has been shared among all of the developers through a common code repository.
- Continuous integration
The developed source code and components have been integrated continuously after the implementation of each of the parts into the overall product. This lead us to have a daily running build on our repository.
- Pair programming
Pair programming has been used when possible. This turned out to be especially useful, mainly when implementing the most critical and complicated parts of the specific user story.
- Customer-on-site
The customer was continuously involved in the development cycle. He/She was the one that approved the user story and verified its correct implementation after each sprint.
- Coding standards
The coding standards introduced by the original developers of RSSOwl have been adopted.

3.1 Aspects of testing

There are two different aspects of testing which have been used for the development of the product.

Acceptance testing

These tests have been written together with the user story and assure the correct implementation of the specific user story from the customer point of view.

Example of the user story “Read statistics pie-chart”

“When the user starts RSSOwl, it can choose from the menu Window>Other>Statistics View. Once the view is open, the user sees at the bottom a pie-chart diagram that visualizes the percentage of read/unread items over the total amount of news-feeds on all bookmarks.”

Unit testing

As mentioned, one of the adopted practices from the XP development methodology has been the test-driven development (TDD). This has been realized by writing unit tests using the jUnit framework and by proceeding in the following way according to the TDD cycle:

1. Create a new test
2. Run all the tests and verify that the newly created test fails
3. “Hack” the code to make the new test pass
4. Run all of the automated tests and see them succeed
5. Refactor your code
6. *(Run all of the tests again to see refactoring has broken nothing)*

Moreover boundary value analysis has been taken into consideration when writing the unit tests, meaning to pay particular attention to test the lower and upper boundary conditions of the tested artifact.

The final RSSOwl product comprises about 376 test runs in total, where about 25 test runs have been written during our involvement.

4 Competitive priorities

The competitive priorities can be divided into cost, quality, time and flexibility. During the development, the following competitive priorities have been excelled:

- Quality: Consistent quality
- Time: On-time delivery, fast delivery and development speed
- Flexibility: customization flexibility

The cost factor has not been considered, since the project has been developed as an open source project, wherefore it was not possible to estimate it in terms of selling cost. However, due to our adopted development process (see previous section) the focus has been more on quality, time and flexibility rather than low cost.

Bringing these competitive priorities together with the mentioned development practices leads to the following situation:

Competitive priority	Adopted XP practice
Consistent quality	Test-driven development, pair programming, planning game, collective code ownership

On-time delivery	Continuous integration, short releases
Fast delivery	Short releases
Development speed	Simple design, refactoring, customer on side
Customization flexibility	Customer on site, simple design

5 Implementation

The following table represents the project backlog and outlines all of the implemented user stories with the according details such as the estimated versus the actual size, the priority, gap and the ID of the sprint during which the story has been implemented.

ID	Title	Estimate (h)	Actual (h)	Priority	Gap	Sprint ID
1	Remove "internal browser" menu entries	41.5	38.25	1	-3.25	sprint 1
2	Create project wiki	1.5	1.5	1	0	sprint 1
3	Adjust browsing preferences	23	20.5	1	-2.5	sprint 2
4	Create label view	25	24	1	-1	sprint 3
5	Labels filtering	26.5	27.5	1	+1	sprint 4
6	Create Statistics View	21	13	1	-8	sprint 5
7	Add read statistics	14	12.5	1	-1.5	sprint 6
8	Statistics-view refresh	12	9	1	-3	sprint 6
9	Read statistics pie-chart	13	14	1	+1	sprint 7
10	Add feed update statistics	7	5.5	1	-1.5	sprint 8
11	Statistics paging	6	7.5	1	+1.5	sprint 8
12	Add context-menu in Labels view	7	6.5	2	-0.5	sprint 9
13	Show Count of Sticky News for Labels	6	6	2	0	sprint 9
14	News item rating	14	14.5	2	+0.5	sprint 10

5.1 Architecture Overview

RSSOwl is structured as a set of interacting plugins (according to the Eclipse plugin architecture) on top of the Eclipse Rich Client Platform (RCP).

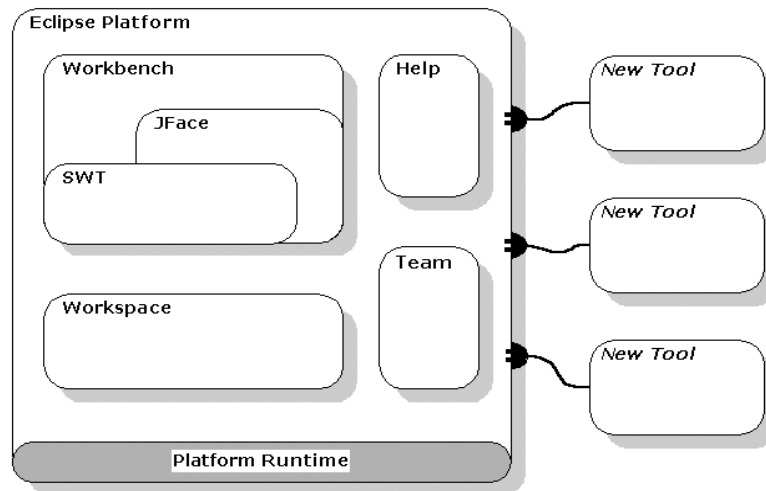


Figure 1: Eclipse plugin architecture
 Source: <http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.html>

The main plugins are the following

- **org.rssowl.core**
 Contains the application's core components such as the domain objects that have to be persisted and the appropriate services for instantiating DB connections.
 It is constructed in layered way, by separating the domain/business logic from the actual persistency logic, using patterns like Separate Interface³ and Data Objects (DAOs):
 - *org.rssowl.core.persist*
 Package containing the interfaces for the different domain objects.
 - *org.rssowl.core.internal.persist*
 Package containing the implementations of the domain objects
 - *org.rssowl.core.persist.dao*
 Package containing the interfaces for the different DAO classes
 - *org.rssowl.core.internal.persist.dao*
 Package containing the concrete implementations of the DAOs.
- **org.rssowl.ui**
 This plugin makes use of the org.rssowl.core plugin for accessing the persistence functionalities and the domain objects. Its main responsibility however is the graphical user interface. It contains the jFace components such as the Editors and Viewer classes with all of the according content provider objects, sorters etc.
 It makes heavy use of jFace Actions (Command objects) for carrying out user-commands (i.e. adding new bookmarks,...) and for interacting with the core plugin (i.e. performing persistence operations...).

5.2 Outline of the core functionalities

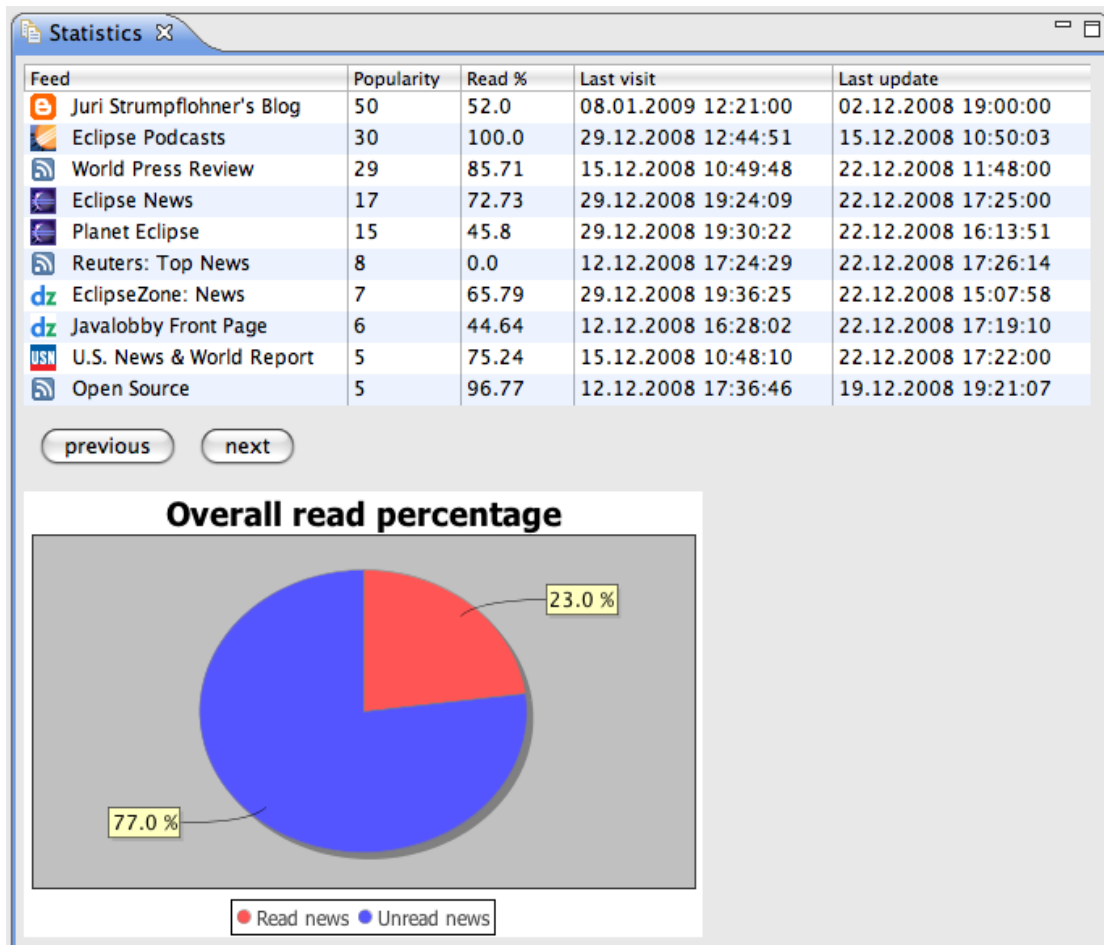
In this sections the implementation of the core functionalities defined in the project mission will be shortly outlined.

³ Martin Fowler (2003), "Patterns of Enterprise Application Architecture", Addison-Wesley

Statistics view

For this user story a new jFace view has been created and integrated into the main application's user interface. The view shows different statistics for the feeds the user is subscribed to. Such statistics include the popularity, read percentage, last visit and last update and are calculated by making use of the attributes of the BookMark and News objects.

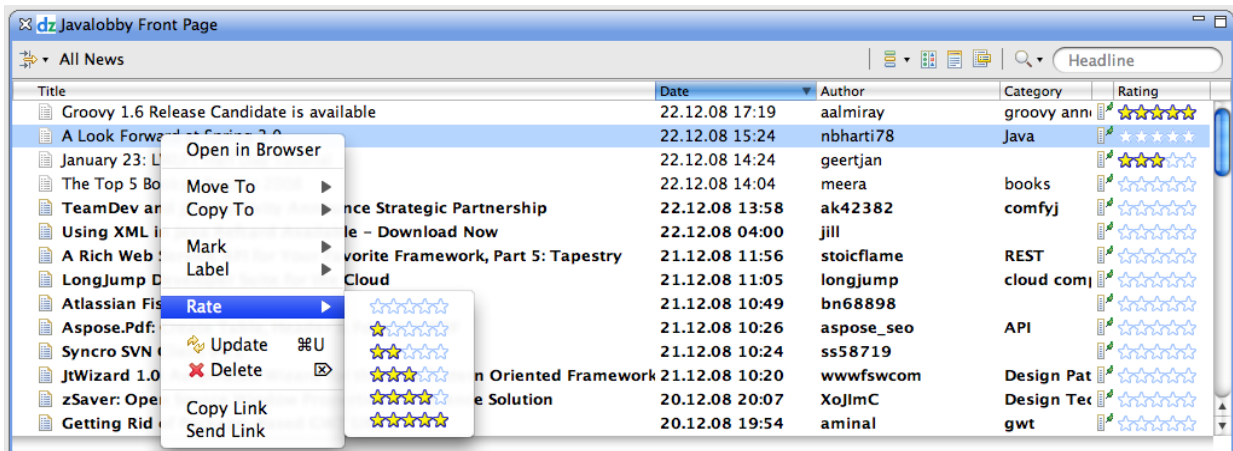
Moreover, later in a separate user story a visualization for the read percentage statistic has been added in form of a pie-chart. This has been achieved by using the JFreeChart⁴ library.



News item rating

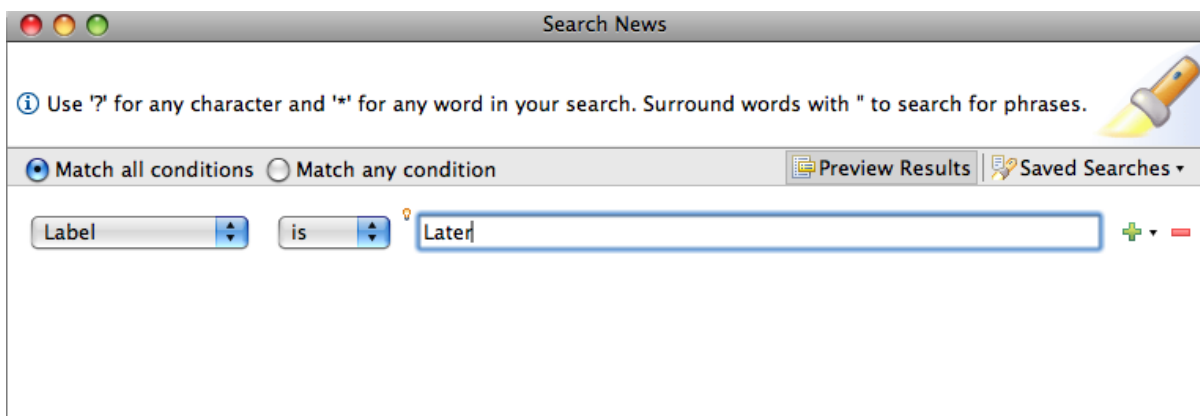
For each news item inside a subscribed news feed, the user rating is stored. The user has the possibility to modify this rating over the context menu by selecting one or more news entries on the main news table of the user interface. The rating has been visualized in terms of star images, which is most commonly used on modern desktop and web applications. Moreover this increases the user friendliness.

⁴ <http://www.jfree.org/jfreechart/>

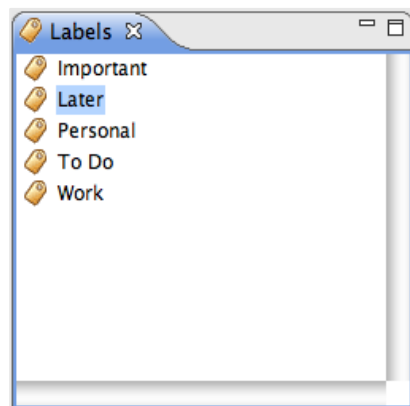


Filtering of news items according to their labels

In the original application, the author has just given the possibility of assigning labels to the news items and to search them over the implemented search functionality (as shown in the picture below).



Since labelling is a very useful functionality for navigating and filtering among the news items, we decided to make this process more straightforward and convenient for the user.



For this purpose a new view extension has been created containing all of the user-defined labels.

Moreover when clicking on the label, all of the associated news items are automatically filtered and displayed on the main news item table of the user interface.

Removal of the internal browser

The original RSSOwl application provided the possibility of opening the read news items directly within the application in its internal browser. Since one of the mission's goal was to remove one of the features of the open source application, we decided to remove the internal browser functionality. The main reason for this decision was due to encountered rendering problems.

So, the affected components have been isolated and removed from the overall product. In addition RSSOwl's preferences have been adjusted and the default behaviour has been altered such that all of the news items are opened by using the user-defined default external browser of the operating system.



6 Metrics

Metric	Before modifications	After modifications	Difference
LOC	67,514	69,204	1,690
# of packages	47	51	4
# of classes	467	493	26
# of methods	3,859	3,978	119
Average CC	2.71	2.71	-
# of test methods	351	376	25
Code coverage (covered instructions)	Total: 58 % org.rssowl.ui: 11.6 % org.rssowl.core: 70.7 %	Total: 59 % org.rssowl.ui: 12.5 % org.rssowl.core: 71 %	1 % 0.9 % 0.3 %