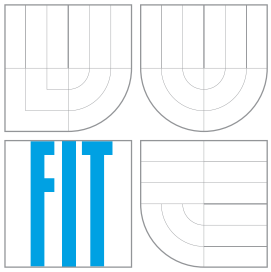
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

INTEGRACE E-MAILOVÉ KOMUNIKACE DO IDE ECLIPSE

REMAIL - INTEGRATING E-MAIL COMMUNICATION IN THE ECLIPSE IDE

SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE
AUTHOR

VÍTĚZSLAV HUMPA

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Ing. TOMÁŠ HRUŠKA, CSc.

BRNO 2011

Abstract

Developers of software systems have to communicate about the project they are building. Especially when working in a distributed development team, such as open source projects, developers must use an asynchronous means of communication. Studies tell us that e-mails are, by far, the means of communication mostly used during the distributed development, opposed to instant messaging, commit comments, or code comments. Therefore, we can imagine archives containing development e-mails enclose essential information concerning various entities of the source code. Unfortunately, such information gets lost with time, since relevant e-mails are hard to retrieve. We have developed REmail, an Eclipse plug-in, to integrate e-mail communication in the IDE. It allows developers to seamlessly handle source code entities and e-mails concerning the source code, without ever exiting from the IDE. Using lightweight linking techniques, REmail retrieves all the e-mails relevant to the chosen source code entities and makes them available to the developer.

Abstrakt

Během vývoje softwaru musí vývojáři mezi sebou komunikovat. Zvláště pokud pracují v distribuovaném prostředí. Například na open source projektech jsou nuceni využít různých asynchronních metod komunikace. Ze studií vyplývá, že ve srovnání s instatními zprávami, komentáři zdrojového kódu, či komentáři verzovacích systémů e-mail představuje zdaleka nejpoužívanější způsob komunikace při distribuovaném vývoji softwaru. Lze si proto představit, že archívy vývojářských e-mailů obsahují podstatné informace o nejrůznějších entitách zdrojového kódu. Časem však se takové informace ztrácejí, jelikož tyto e-maily je těžké dohledat. Proto jsme vyvinuli REmail, zásuvný modul pro Eclise, integrující e-mailovou komunikaci do IDE. Umožňuje vývojářům pracovat souběžně se zdrojovým kódem a e-maily, které jej diskutují, bez nutnosti opuštění IDE. Využitím relativně výpočetně nenáročných technik REmail dohledá všechny e-maily relevantní k vybrané entitě zdrojového kódu a umožní vývojáři s nimi pracovat.

Keywords

e-mail to source code traceability, inter-developer communication, IDE integration, Eclipse, plug-in development

Klíčová slova

vazba mezi e-maily a zdrojovým kódem, komunikace mezi vývojáři, integrace do IDE, Eclipse, vývoj zásuvných modulů

Citace

Vítězslav Humpa: REmail - Integrating e-mail Communication in the Eclipse IDE, semestrální projekt, Brno, FIT VUT v Brně, 2011

REmail - Integrating e-mail Communication in the Eclipse IDE

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně pod vedením pana prof. Tomáše Hrušky.

.....
Vítězslav Humpa
January 6, 2011

Poděkování

I would like to give thanks to prof. Tomáš Hruška, prof. Michele Lanza and Alberto Bacchelli for the supervison and overall leadership while working on REmail project.

Contents

Chapter 1

Introduction

Nowadays, when creating a software system, developers spend a significant amount of time inside an Integrated Development Environment (IDE). Unless they work on a small project of their own, developers are often part of a team that works on the same project.

Since developers work on IDEs that are not connected one to another, they must find alternative ways for **communicating** ideas and synchronizing work. There are many ways of communication between them. In addition to face-to-face meetings developers often communicate through instant messaging, notes inside commits, commenting the source code, posting bug reports, or they can post **e-mails** inside **mailing lists** [4].

Face-to-face meetings are the preferred method of communication when developers work in a collocated team. However in the development of large **open source** systems, developers might be spread all over world, thus making frequent face-to-face meetings difficult, if not impossible to organize.

Studies report[9], that in these cases, e-mails are the most widely used means of communication among software developers. For large projects, various mailing list are usually established to allow information exchange. Nevertheless, as the name suggests, development[1] lists are those that are most important for developers.

Such mailing lists are used to communicate about various programming issues that raise during development. Therefore, these archives contain e-mails that are an important source of **information** about high-level design decisions as well as low level implementation concerns and developers' social structure[5].

Programmers use applications that are external to the IDE to handle the content of mailing lists: No matter how related these e-mails are to software development, these are completely **disconnected** from the IDE. Such situation forces developers to interrupt the programming flow in the IDE whenever they need to operate with them. As there are no means of linking the contents of e-mails to the source code entities they refer to, important information gets **lost** with time.

1.1 Goal of this project

Currently, Bacchelli et al. are conducting a research, that aims at devising lightweight methods to recover the traceability **link** between source code artifacts and e-mails [2][5]. Finding e-mails that are actually relevant to code entities is not trivial, mostly due to complicated nature of informal human communication. Bacchelli et al. provided a number of

[1] Those often marked with a *dev* or *devel* keyword

source-to-e-mail linking methods based on regular expressions (thus lightweight), providing different levels of precision and recall.

The goal of our project is to create **REmail** - a plug-in, which integrates e-mail communication into the Eclipse IDE. It enables developers, by taking advantage of these lightweight methods to exploit the information that can be obtained by the process of linking the e-mails with code entities.

1.2 Structure of the Document

In Chapter 2, we discuss in detail the underlying research that led to REmail. We also analyze related work.

In Chapter 3 we introduce REmail: We describe the evolution of the project as well as its current implementation, and we provide instructions on how to work with it.

In Chapter 4 we present a use case based on the experience of using REmail on the *Freenet* project during the creation of REmail.

In Chapter 5 we conclude the thesis, summarize its results and discuss possibilities of future improvements.

Chapter 2

Background research

Purpose of this chapter is to introduce the research that led to creation of REmail.

Topics presented in both sections of this chapter form a base, from which came the idea of REmail. In the Section 2.1 we discuss inter-developer communication relations and rationalize benefits of REmail[4], while in the Section 2.2 we present techniques that form logical core of the plug-in[2].

2.1 Communication between developers

Especially when working in a distributed development team, such as open source projects, developers of software systems have to communicate about the project they are building. They use various means of communication to do so.

Posting e-mails into mailing lists is by far the most spread way of communicating about development. In open source environment, nevertheless face-to-face consultations remain the most popular in collocated teams[9].

Face-to-face meetings are considered the most effective form of communication among collocated developers [4]. Developers loose no time communicating their ideas or describing problems thanks to the use of spoken language. There can be misinterpretation or misunderstanding as in case of written communication, but they can be resolved immediately [9]. However face-to-face meetings are not without drawbacks.

Presence of other developers is distracting. Questions make developer loose focus on his task, which he needs to regain later. As opposed to electronic means of communication, such spoken conversation itself is not stored. Developers tend to create mental models of the system [9]. If they envision such a model by talking with other developers, they are likely to loose portion of the model comprehension with passing time, unless they write down well organized notes.

Finally, face-to-face communication is geographically limited, which, especially in open source community, makes extremely hard to organize it in necessary frequency.

There are many methods of electronic communication that are used for discussing the code-related topics. They stand in between face-to-face and e-mail communication when it comes to their popularity and spread. We introduce them in following list. At the end of this section, we explain why we favor e-mail communication for IDE integration.

- **Design documents** provide a high level model of the system and are useful for understanding of the software system and its decompositions. Usually developers start creating them before commencing coding tasks. Therefore design documents, in

both graphical and textual form, serve as the point of rationalizing the structure both for first-hand developers as well as ones that join the project later and need to master the system architecture structure.

On the other hand, since being produced ahead, design documents often fail to keep track of the project evolution and get less precise in relation to source code with time. In addition, design documents mostly describe only a high level view of the system. For these reasons, it is often difficult to find the traceability link between actual source code and design documents [10][1].

- **Code comments**, as an opposite to design documents, are tightly related to the lowest level model of the system, the source code itself. This means that information contained in them cannot be used to describe higher-level relations between artifacts.

  They are already linked to all of the code artifacts that they annotate, since they are part of the source code itself. This induces code commits simply for updating them. They also leave no space for discussion.

- **Commit comments** are already linked to the source code as well. In relation to code comments, commit comments have broader focus as they can mark out changes in more than just single file. In this way they can give insight into higher-level composition of the system.

  In practice however, they are mostly used to inform about fixes, or they are sketchy reports of new features and are very short. Because they are attached to code updates, it is not possible to reply on them and therefore start a discussion. (In practice, this is often overcome as modern version control systems post commit messages into mailing lists. Actually this has further repercussions in usage of REmail, more in Chapter 4.)

- **Issue reports** and bug tracking system are nowadays necessary parts of any software project. They allow for filing textual reports containing useful information related to the source code.

  The spectrum of this information is quite broad, since issue reports are not only filled by developers, but also by other interested groups - notably beta-testers and users.

  In modern bug tracking and project administering systems[1], these reports can be commented and replied to in threaded way.

  Links between these reports and source code could be established by using heuristic and pattern matching methods similar to those introduced in the next section [6]. However, the communication in bug reports is focused on the issue itself and it hardly crosses these borders. It is uncommon to find high level concerns or design rationale discussed in this media [4].

  As a confirmation about the reduced focus of bug reports, new developers are commonly employed to work on bug fixing, as it requires less high-level architecture than implementing new features [9].

- **Instant messaging** is close to face-to-face meetings. Thanks to the nature of modern real-time protocols[2], developers communicate as in real life, except that they are using textual means instead of natural languages.

[1] E.g. Bugzilla, Mantis or Jira
[2] E.g. Internet Relay Client (IRC), XMPP, ICQ, etc.

It is not used frequently in collocated teams, where developers can meet face-to-face, but is often employed inside various open source communities, where it provides means for rapid coordination and for conducting online meetings to discuss various development issues [14][9][8]. Responses on instant messages are expected immediately, which is usually the case. Also developers can have parallel IM conversations with many counterparts.

However, similarly to face-to-face, IM has disadvantages. Particularly the synchronous aspect of instant messages raises the issue of interruption and loss of concentration. As with all synchronized means of communication, problems of dialogue between people in different time zones emerges as well.

Internet telephony. With introduction of applications[3] for conducting online voice conversation, internet-telephony seems to be taking off as a new way of communication among developers.

Unlike when using telephones, it is still closely related to instant messaging. We presume that developers prefer to use IM for code related issues, however needs arise to communicate about specific and complex issues. In that case, use of internet telephony and recently also audio-video transfers saves time.

This method of communication is inherently close to face-to-face meeting and brings its advantages and issues with it. Notably, as conversations are usually not being recorded, linking them to source code artifacts is not possible.

Developers reported that responses to questions sent by e-mail can take hours or even days be received. Original questions are often misunderstood by readers who do not provide correct answer to the matter at hand. And developers generally consider writing e-mails to be tiresome [4][9].

These problems are also common in other communication means, especially in design documents, issue reports and partially in IM. However e-mails offer advantages that make them a preferred method for inter-developer communication.

First of all, e-mails are used to discuss issues about any level of abstraction - from low level implementation details up to high level design decisions. They can be written by anybody who has subscribed into a mailing list, not just by developers themselves (which is a problem especially with code and commit comments), but also by beta-testers, and, consequentially end users of the system.

Therefore, we can often link e-mails to any source code entity that has been discussed at some point. In addition, e-mails offer the additional information stored in the headers (defined by the RFC5322[11] message format) and when used in context of mailing list, thread information is also available. Such information helps the effort of code-to-mail linking that can be done using techniques presented in section 2.2.

Finally, the code-to-e-mail linking can be used to study a project evolution. This is useful when attempting to reverse engineer a system, with the goal of understanding its parts by (usually new) developers tasked to add new features or fix bugs.

For all these reasons, we consider **e-mails** as one of the best candidates to improve the communication between developers. Our goal is to support in a seamless way e-mail communication within the IDE. We are in process of accomplishing this by creating **REmail**.

[3] E.g. Skype and Google Talk

2.2 E-mail-to-code linking techniques

Since 2009, Bacchelli *et al.* have been carrying out research aiming at devising methods of linking different entities of source code to e-mails that refer to them [2][5]. Bacchelli *et al.* have experimented with a variety of different methods to do so. The goal is devising lightweight approaches, which could be applied on a great number of e-mails in a reasonable amount of time (that is in seconds). They achieved that criteria using regular expressions.

Bacchelli et al. began with simple intuitive techniques (e.g. simple classname search) and as results were collected from experimenting on sample mailing lists, more complex approaches were devised. Finally, a number of lightweight matching methods were devised[2], each one giving results with various accuracy. Such lightweight methods have been implemented in REmail, giving it its core functionality.

These are the six methods currently implemented by REmail, among which developer can choose:

1. **Class name, case insensitive** - Usually, when being referred to in e-mails, classes are simply mentioned by their names. Thus, the first method simply searches for the classname inside the text of e-mails. Developers don't always write with proper capitalization, so this method is not case sensitive. Also, classnames can appear at the end of sentences, followed by "." or they can be put inside quotes, therefore no restrictions were imposed on the characters surrounding them.

2. **Class name, case sensitive** - Results of search using previous method have a reasonably high recall[4] value, however the precision[5] is low. Since it is a common practice to name classes in a way known as CamelCasing[6] the second method utilize case sensitivity to improve precision.

3. **Strict regular expression, case insensitive** - Results of case sensitive search are considerably more precise than in the first approach, however still about one third of relevant e-mails are recovered in average. A complex *strict* regular expression (Figure 2.1) was constructed to get close to upper bound of precision.

```
(.*)
(\s*)
(<beginning of package>)?
(.|\\|/|\s)
<last part of package>
(\.|\\|/)
<class name>
(.java|.class|\s+)
(.*)
```

Figure 2.1: Strict regular expression as implemented in REmail

In this approach source code file extensions and package fragments are taken into account, the last part of package name is required. Since such regular expression is quite rigorous, case sensitivity is not observed.

[4] The percentage of relevant e-mails actually retrieved, on all the relevant e-mails existing
[5] Many irrelevant e-mails were also linked
[6] E.g. *ArchiveManager* or *MailContentView*

4. **Loose regular expression, case sensitive** - Strict regular expression approach indeed brings precision to a high number[7]. However the recall is minimal[8]. Thus the strictest criterion requiring the presence of last part of package was dropped. Presence of quotation marks next to the name was allowed as well.

```
(.*)
(\s*)
(<package>)?
(.|\\|/)
<class name>
(.java|.class|\s+|,,|,)
(.*)
```

Figure 2.2: Loose regular expression as implemented in REmail

5. **Mixed, using dictionary, case Sensitive** - If we take into account the strict method and the method using just a classname for search, they score opposite in precision and recall. *Case sensitive* method has a high recall value and low precision, while *Strict regular expression* approach gives opposite results. Idea came to combine the two, trying to get the best of both methods.

   It was presumed that the small precision of the simple name matching method is caused by worlds that are classnames, but that also exist in standard natural language, e.g. names like *Bookmark* or *Cookie*. Strings like *ConfigToadlet* are not a part of that, thus are likely to be classnames when referred to. Therefore in this combined method, English dictionary is queried for the classname. If the string is present in dictionary, the *Strict regular expression* method is applied, otherwise the *case sensitive* search is conducted.

6. **Mixed, using CamelCase, case sensitive** - Technique using dictionary provided results with both precision and recall above fifty percent. However, inspecting a dictionary is time consuming, thus the idea to use *CamelCasing*, instead of dictionary was devised. When the string in hand is in *CamelCase*, (e.g. *MailView* or *SelectionProcessor*) simple case sensitive search is run, while otherwise, (e.g. *Bookmark*, *Cookie*) Strict regular expression matching is conducted. While having the same recall as the previous, this method had actually increased precision by few percent, while being considerably faster and less memory dependent.

Out of all these methods, it would seem that the *CameCase* search is the best in most situations, when the search is applied on variety of classes (or if not on the entire project) in the same time. However, the other methods are also useful when the developer needs to achieve the highest possible precision, or recall, in cost of the other measure.

[7] 94% of all relevant e-mails in *ArgoUML* test case
[8] Only 10% of relevant e-mails were present

Chapter 3

REmail

In this chapter, we detail REmail, an Eclipse plug-in we have created to put the ideas previously presented into practice.

First, we introduce Eclipse itself, we get acquainted with its structure and the basics of plug-in development. Then, in section 3.2, we tell the story of REmail and we describe how it has evolved since we have started working on it. Following that, we explain the state of the current implementation of REmail, detailing the how various parts of the plug-in are realized (3.3). Finally, the last section is the user manual, with goal of explaining how to install, setup, and use REmail in practice. (3.4)

3.1 Eclipse

With the goal of putting the idea of e-mail integration into practice, we had to decide what IDE we would actually use. There are many choices, especially if we consider the various development environments for different programming languages.

The idea was certainly to make REmail work in an IDE that can be used for developing systems in multiple languages. Since the linking techniques were shown to work with many programming languages [5], therefore many environments, like those centered on C/C++ were put out of consideration.

Since we wanted to build a plug-in, we were also looking for an environment with a good modular structure. NetBeans[1] and Eclipse[2] - Java based development platforms that also allow programming in other languages - thus came into consideration.

Eclipse is currently the most widely used IDE in the open source community, and also provides a considerable amount of documentation and tools for plug-in development. Having a reasonable modular structure, we have decided to build REmail as a plug-in of Eclipse. We are also considering creating a version of REmail for an alternative IDE, especially NetBeans.

3.1.1 Structure

The main part of Eclipse for both user interaction and plug-in development is *workbench*: the main window that always has one of the *perspectives* active. By *perspective* we mean

[1] www.netbeans.org
[2] www.eclipse.org

the setting that decides how the important parts of *workbench* are spread over the window. Plug-ins can access active editors and views through the *workbench*, and also open and manipulate their own [13].

The *workbench* contains two main types of windows, placed into subcontainers, that most UI contributing plug-ins work with: *views* and *editors*.

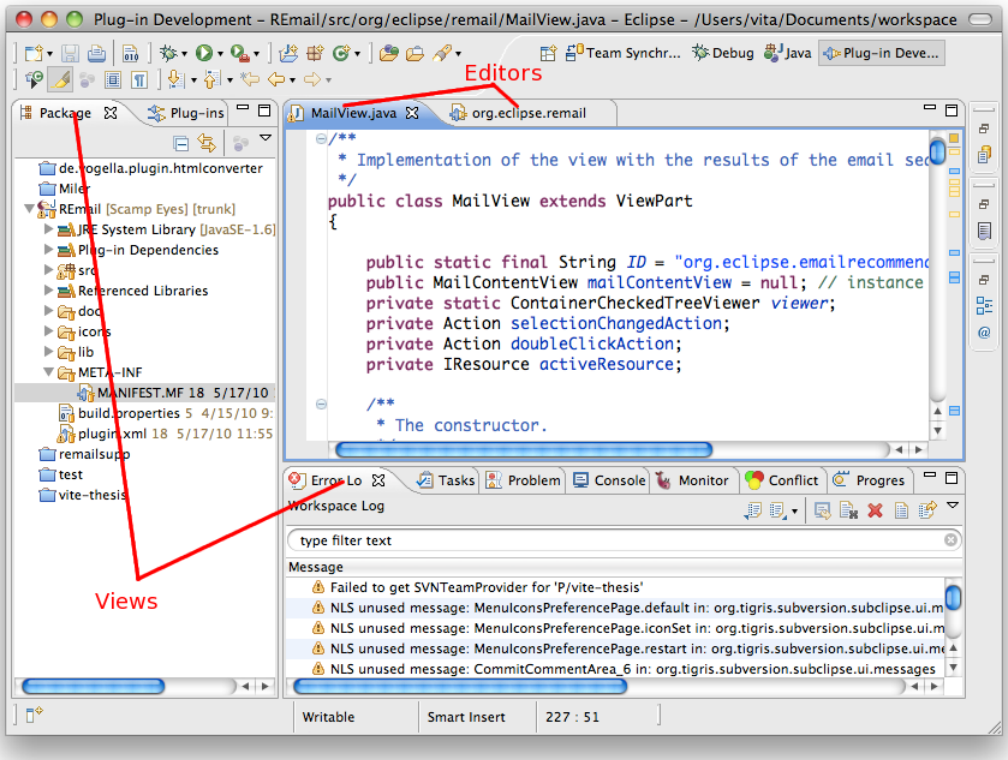


Figure 3.1: The Eclipse IDE

Views, such as *Package Explorer* or *Mail View* and *Main Content View* in REmail, allow plug-ins to display or edit important information. The information is commonly viewed using one of the JFace[3] controls based on tree, list or table.

Editors edit a certain resource, often a file. There are different types of editors in any Eclipse distribution: when opening a file for editing, the proper type of editor is chosen based on file extension. Eclipse editor framework allows plug-in developers to create all aspects of a typical IDE editor on their own, including autocompletion menus, hovers and syntax coloring. Some of the most common generic editors (like default Java editor) can also be extended by plug-ins.

3.1.2 Basics of plug-in development

Any Eclipse distribution is a collection of different plug-ins that together form a desired environment. Unlike some other IDE's, Eclipse itself is actually a small core, that loads all

[3] A set of advanced widgets based on Standard Widget Toolkit offered by Eclipse project (http://wiki.eclipse.org/index.php/JFace)

present plug-ins into place: All we can see when using Eclipse are plug-ins [7]. In a clean installation most of the plug-ins are part of core *org.eclipse* package.

Every plug-in extends the functionality of Eclipse by connecting itself into one or more *extension points* that are defined by other plug-ins. The same plug-in can also define its own *extension points* to provide information to other plug-ins.

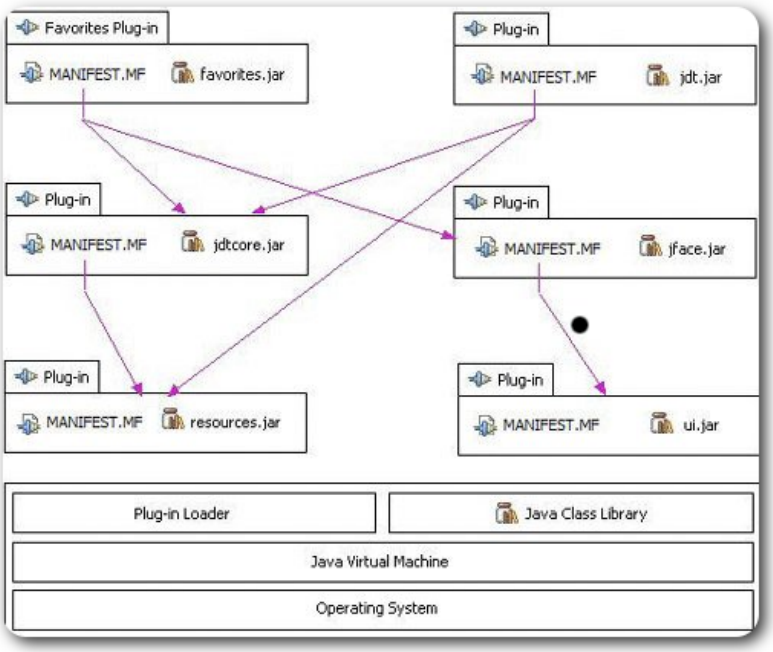


Figure 3.2: Eclipse as a collection of plug-ins [13]

Hence all the plug-ins that are part of Eclipse define a *MANIFEST.MF*, and *plugin.xml* - files declaring, amongst other things, what plug-in bundles are used by the plug-in, what extension points are needed, and what other points are defined.

The following is the MANIFEST.MF of REmail. Especially *Require-Bundle* is needed to tell Eclipse the basic plug-ins required by REmail.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: REmail
Bundle-SymbolicName: org.eclipse.remail;singleton:=true
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: org.eclipse.remail.Activator
Require-Bundle: org.eclipse.ui,
 org.eclipse.core.runtime,
 org.eclipse.jdt.core;bundle-version="3.5.1,,,
 org.eclipse.core.resources;bundle-version="3.5.1,,,
 org.eclipse.jface.text;bundle-version="3.5.1,,,
 org.eclipse.ui.ide;bundle-version="3.5.1,,,
 org.eclipse.ui.editors;bundle-version="3.5.0,,,
 org.eclipse.jdt.ui
```

```
Bundle-ActivationPolicy: lazy
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Bundle-ClassPath: lib/postgresql-8.4-701.jdbc4.jar,
 lib/sqlitejdbc-v056.jar,
 lib/commons-lang-2.4.jar,
```

Notice the line *Bundle-ActivationPolicy: lazy*. Eclipse by default only preloads the plug-ins defined by their manifests. The extension is fully loaded and all resources allocated at the time of actual usage of plug-in, by the time we open some of its views, or execute a plug-in-added command in the menu.

Plugin.xml

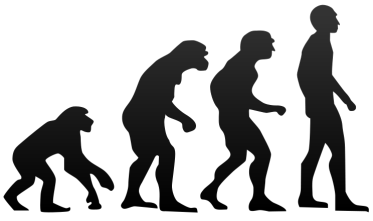
Plugin.xml is an essential file, defined by every plug-in, that allows developers to specify in XML format any extension point that is needed, or defined, by the plug-in. Every extension point comes with mandatory and optional arguments, which describe the plug-in integration precisely.

Points of extension are added as <*extension*> tags with the arguments added as sub-tags. An example describing the *Mail View* of REmail follows:

```
<extension point="org.eclipse.ui.views,,>
    <view
          class="org.eclipse.remail.MailView,,
          icon="icons/sample.gif,,
          id="org.eclipse.emailrecommender.MailView,,
          name="E-mails,,
          restorable="true,,>
    </view>
 </extension>
```

The java class implementing the specified functionality is pointed out. For any extension point in REmail, a class needs to implement particular interface provided by Eclipse, as depicted in its documentation.

3.2 The evolution of REmail



Idea of creating REmail came as logical step after the research on Lightweight source code-to-e-mail linking techniques conducted by Bacchelli *et al.* in 2009 [2]. The initial idea was very simple: To have an integration into an IDE, that would let the user select a class in that environment, conduct a search based on the preferred linking technique, show the results and let the user read selected e-mails in a convenient way.

The project started as a part of *Software Design and Evolution* at University of Lugano. Initially a deliverable as a part of SDE project had to be negotiated. At that point the looks and inner workings of REmail started to take shape. First of all Eclipse was chosen as an IDE to put REmail into: Not only because of its structure and available documentation, as several members of REVEAL research group had positive experience extending Eclipse, notably in case of the *Syde*[4] project.

3.2.1 Early stages

The main goal of the first phase of REmail development, has been to implement the lightweight linking techniques and if possible present them in extended user interface of Eclipse. This was done, and a basic version of REmail was implemented (see Figure 3.3).

As a part of their research, Bacchelli *et al.* created a crawler based on Miler[3] capable of fetching a mailing list of any project present on *markmail.org*[5] into a PostgreSQL database. Some of the mailing list were already present in that form, therefore we decided to use PostgreSQL as a means of e-mail storage for the plug-in. As a well established database system, PSQL also proved as excellent platform for implementing linking methods, since they are mostly based on regular expressions.

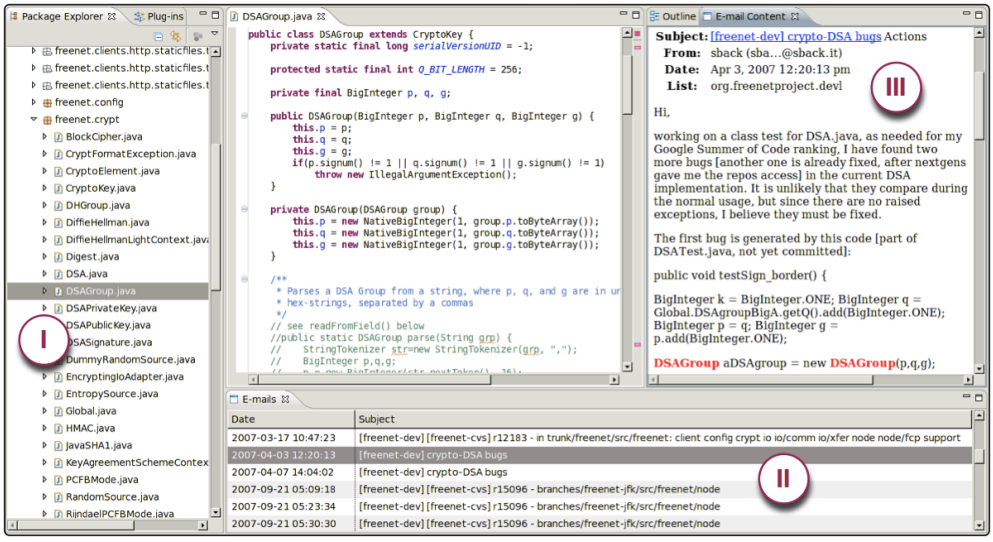


Figure 3.3: Initial implementation of REmail

In first version of REmail, the search was fully connected to the actual UI of the plug-in and coded together. SQL SELECT statements were used to obtain data using regular expression support of PSQL and a proper JDBC driver. This version did not have any preferences implemented yet and every setting was hardcoded.

Developer could select a single class in the package explorer and select REmail search from the context menu showed at (I). The results of the linking were displayed in the table (II) and after selecting a message, its text appeared in a browser shown at (III). This layout proved to be effective, and with many additions to the UI usability and functionality, is still a basic preferred layout in the current implementation.

[4] http://syde.inf.usi.ch/

[5] A free service archiving mailing lists of various open source projects

3.2.2 Switching to MBox

Using PostgreSQL proved to be fast and easy to implement, but it is not without drawbacks. First of all, an instance of PSQL server has to be ran for the plug-in. That might be useful in some cases, as it enables multiple developers to work simultaneously with a single DB (For example, we can imagine employees in a company setting up a single server for all interested members of the team). Another advantage of this centralization is that, when removing ill-linked e-mails from the database by one developer, these e-mails will no longer show up for also the others. On the other hand this could be dangerous too, since not just ill-linked e-mails can be removed: what is uninteresting for one developer might be important for others working on different parts of the system.

However, we want REmail to be also useful for single developers trying to understand, or develop, existing system. Therefore we were looking for an integrated solution that would not require an external software to be ran, other than Eclipse.

Since we are working with e-mails we thought about taking advantage of an established e-mail client. Thunderbird[6] is widely spread e-mail client both in industrial and open source environment, thus we started exploring the possibility of using it as the source of e-mails. Any developer, subscribed to a mailing list in question, would have e-mails of the list stored in Thunderbird, for REmail to utilize. We thought about a non-invasive of searching through the emails: two possibilities came into our attention.

A plug-in for Thunderbird

Idea is to create a separate plug-in for Thunderbird, that would feed data to REmail. In that way, both Eclipse and Thunderbird would have to be running simultaneously. The actual search would be done by Thunderbird extension, that would then feed the results to Eclipse. We wanted to use Unix sockets as channels for the interprocess communication. In this way, the possibility of Thunderbird running on different machine was open, giving the green light to centralization like in the PSQL solution.

In contrast to already existing PSQL search implementation, this solution would only give benefit in better e-mail management for the user. The biggest drawback is, that this solution would be bound only to Thunderbird client. While this can be non problematic for many developers, others use different clients, with old mailing list content already loaded into them. In addition, we had difficulties implementing Thunderbird plug-in according to our needs, so we decided not to follow this course of action.

Accessing MBox data

The second approach (which we actually implemented) is to get e-mail data directly from the filesystem. Thunderbird uses a *MBox*[7] format to store e-mail data: All the messages from single Thunderbird folder, (e.g. *inbox* or *sent*) are stored as a plain text in a single file. They are basically stored in their original format similar to RFC822[x]. Any additions by the email client are added as a header starting with *"X-"* (e.g. *"X-Mozilla:"*).

[6] http://www.mozillamessaging.com/thunderbird

[7] MBox is a generic term for a family of related file formats used for holding collections of electronic mail messages. All messages in an MBox mailbox are concatenated and stored as plain text in a single file. The beginning of each message is indicated by a line whose first five characters consist of "From„ followed by a space and the return path e-mail address. A blank line is appended to the end of each message. - Wikipedia 05/2010

If the user imports an entire mailing list into a single folder in Thunderbird (separated from other emails...), all such e-mails can then be accessed by REmail in the corresponding file. Notably, this format is used by many other clients like KMail, the Mac OS X Mail application, Eudora, etc. This gives the developer a broader choice on the e-mail client. The search module for MBox we have implemented in the REmail works with general MBox format, without any dependence on a particular client.

3.2.3 Threads of e-mails

In mailing lists, e-mails are usually organized into threads. Some of the e-mails are standalone (mostly announces), but whenever there is a reply on some topic that starts a discussion, subsequent e-mails are handled in threads by e-mail clients. We needed to implement such feature into REmail as well, to provide necessary orientation between results of the search. At same time we wanted to provide an automated tool that would be able fetch a complete desired mailing list of project for the user. We presume that developer often does not actually have to be subscribed to the mailing list himself. Additionally, if he is subscribed, there is no common method of getting e-mails that were posted to the mailing list before his subscription. We have devised a way to take care of both of these needs in the same time.

We re-wrote the original Miler based web crawler[3] into Java. This allowed us to modify it so that any *markmail.org* mailing list could be stored into an MBox compatible file. We create a file that can subsequently be imported and manipulated by user's e-mail client. For example in the case of Thunderbird, all that is needed is for the file to be stored inside"Local Folder" directory in Thunderbird's directory structure. After the next launch, the new e-mails will be visible, open for manipulation, and ready to be utilized.

3.2.4 Making it modular

We did not decide to switch completely from PSQL source to MBox. Instead we wanted to have both implemented together, so that they could be both chosen and have the benefits of both of them. In the original implementation, the actual search was implemented together with the UI part of plug-in. For example, a class was invoked as a result of search command, it called PSQL search specifically and then took care of updating the graphical elements of the plug-in. Now with adding a new way of searching, we had to make the plug-in modular, and separate the program logic from the user interface. This was accomplished by creating a general search interface, that is implemented by different classes providing the same linking functionality by separate means (About this in the next section).

3.2.5 Adding features

With the MBox search implemented and REmail structure reworked, we could focus on improving the functionality of REmail, especially the usability.

Preferences

First of all, a preference panel had to be built. We have added a set of properties panes, into Eclipse's Preferences, necessary to setup all aspects of the plug-in. Notably log-in information for PSQL, locations of MBox source files for projects, selection of the e-mail

source and the choice of a lightweight linking method to use. We have also decided to implement message filtering that would give developer a way of refining search results.

Initially the only way to conduct a search was by clicking on the classname in Package Explorer (Figure 3.2.5) and selecting a method of search. This had to be done for every class individually. In current implementation, method selection has a general setting inside preferences.

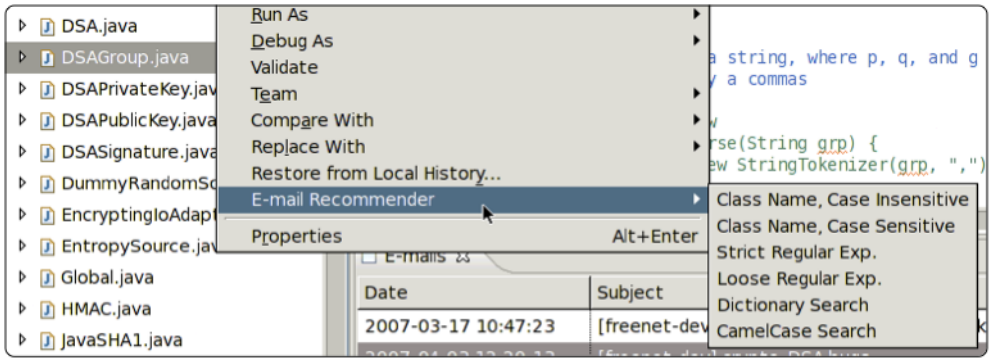


Figure 3.4: Initial way of conducting search

Result indexing

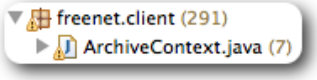
In the first implementation, a search was conducted at run-time, taking a few seconds[8] to produce results. This was hardly an optimal solution, so we devised an indexing process.

Utilizing SQLite JDBC driver[9], we have developed a database based indexing of search. When a search is conducted on any class, the results are stored into the small SQLite databases. By the next time, just selecting a class in Package explorer shows the outcome of the last search instantly. A new search would updat the indexed data.

Package Explorer

Together with indexing came ideas of substantially changing the way the search is conducted and the results presented inside the Package Explorer. Foremost, now it is possible to make a search on multiple classes, packages or even an entire project. By selecting multiple classes, user can start the linking process and index the data for all of them. Similarly, a developer can select a set of packages, in which case the search will be done on all classes included. By running the process on the root of the project, all the classes are linked with e-mails and results stored in the SQLite database file.

Besides that, we have devised a decoration of Package Explorer entities to notify the user if a particular entity has been searched for. We show a number of "hits" each entity has within the mailing list. In the case of a class, we show the number of e-mails linked with it using the preferred method. With package, we show the number of e-mails linked to all of its classes.



[8] When working with Freenet development mailing list with around 22000 e-mails
[9] http://www.zentus.com/sqlitejdbc/

When conducting a search on a bigger number of classes at once, time necessary to complete the linking process can become long[10]. Thus we have made a progress-bar that shows how many classes out of total have already been linked.

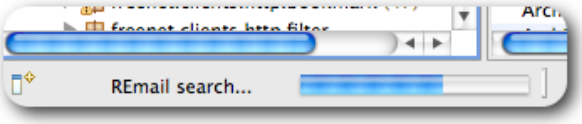


Figure 3.5: Search process progress bar

Viewing the results of linking

The "E-mails" view has evolved considerably. In its first implementation (Figure 3.3), it simply contained a table presenting results using a couple columns. By putting threading of e-mails into practice came the need to show this additional information. Therefore, we modified the view to show data using a *tree viewer* component of JFace.

Figure 3.6: E-mail view using tree viewer

Still, we felt the information was not presented in a sufficiently clean manner. As a next improvement, we joined the previous table and tree viewers together using a view that brings columns into a tree structure, thus providing a better visual presentation.

One of the latest additions at the time of writing this thesis is making it possible for the users to mark emails that are of little or no importance to the class in the question. For this, we modified the *viewer* to display a checkbox that allows the user to express lack of interest in particular e-mail or thread.

E-mail content view

In REmail, using *E-mail content view*, you can read the text of any linked e-mail. At first, we used the html component of SWT, and simply inputed the text of the email inside (*III* in Figure 3.3). This only provided a better text rendering. However, we also felt the need to highlight the name of the linked class: During the usage, the linked class was often only mentioned as a part of a dump of paths, or changes in commits, thus it was difficult to spot. With the classname highlighted a user can quickly see whether this email is relevant.

Another issue to readability of the e-mail is the presence of text from previous messages in the thread. In our test case on Freenet, this was actually the case for the vast majority of the results. The standard way of highlighting the previous *RE:* messages is by putting a

[10] More in chapter 4

number of ">" symbols at the beginning of each line. In the *E-mail content view* though, as text is usually getting wrapped, this was not easy to comprehend visually.

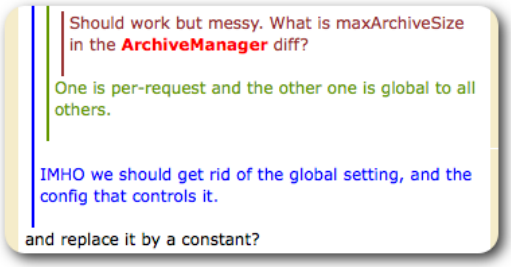


Figure 3.7: Marking threaded conversation in the text of e-mail

Inspired by *RE:* message highlighting in Thunderbird, and a general style of displaying messages on *markmail.org*, we changed the look (Figure 3.7) of this view. Now the text from previous e-mails is preceded by a number of differently colored lines and presented in a different font color.

Editor improvements

Most of the time that developers spend using Eclipse and other similar IDEs, is focused on the Editors, where they actually work with the source code. Often enough (while working with smaller monitors, laptops etc.), they might maximize the editor to completely fill the screen. By doing so, REmail's views are invisible. Therefore we wanted to contribute into the editors of Eclipse itself to provide some support in this situation too.

Text hovers Our original idea was to extend the Eclipse's Javadoc hover. It appears while going over some keyword with a mouse pointer. After researching how to implement this feature, we found that this would require cloning and creating a special editor to do so. Since we wanted to have this feature present in the default editors of any Eclipse distribution, we decided to search for a different method of editor integration.

Side bar ruler markers Markers point out general point of interest in any of the resource files. Known markers present in Eclipse are *problem markers*, *warning markers*, *to-do markers* and *breakpoint markers*, all of which appear when necessary on the left side of editor.

There is also a *bookmark* marker that allows developers to point out places of interest of their own. We have used these markers to provide information about all the classnames visible in the editor's source code, to which some e-mails have been linked. Their appearance can be triggered by a toolbar button, so that user can decide whether to show them. All can also be browsed in the built-in bookmarks menu, which provides another place for developers to check whether some e-mails are linked to the entities existing in the files he works on.

In this subsection we have described a number of features that we implemented in the current version of REmail. We have plans and ideas for future, which will be discussed in 5.2.

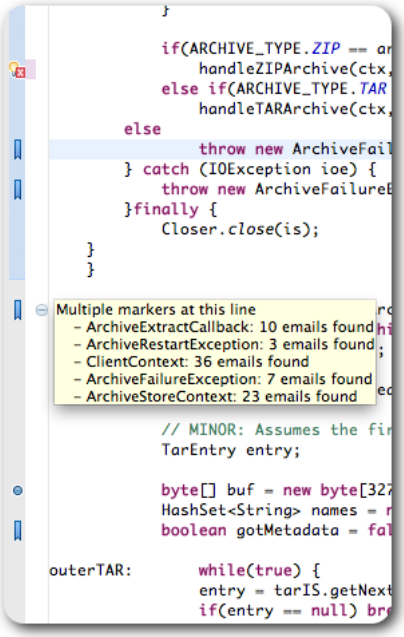


Figure 3.8: Using markers

3.3 Implementation of REmail

In this section, we detail the *current* implementation of REmail. As all the software evolves, and we hope REmail to keep evolving, the content of this chapter will be probably valid only for the current release.

3.3.1 The general structure

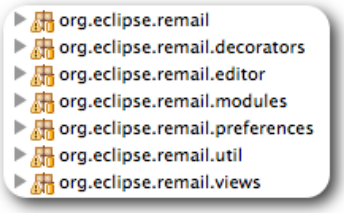


Figure 3.9: Packages of REmail

REmail is implemented in 29 classes, split inside 7 packages. Important parts of REmail are also metafiles introduced in 3.1.2, in which all the parts of Eclipse that are extended by the plug-in are declared, and the classes implementing particular extension defined.

The logical base of the plug-in is the implementation of the lightweight linking methods providing different search strategies. This is the practical core of the plug-in and it has been implemented in a modular way, so that different sources of data could be used for a search.

As we can see on figure 3.10, we created the interface *MailSearch*, which defines the

methods that need to be implemented by any data-source solution. Inputs of these methods are given by the particular needs of the linking method. In case of simple case sensitive and insensitive searches we need to input a classname, while for other methods also complete package name is required.

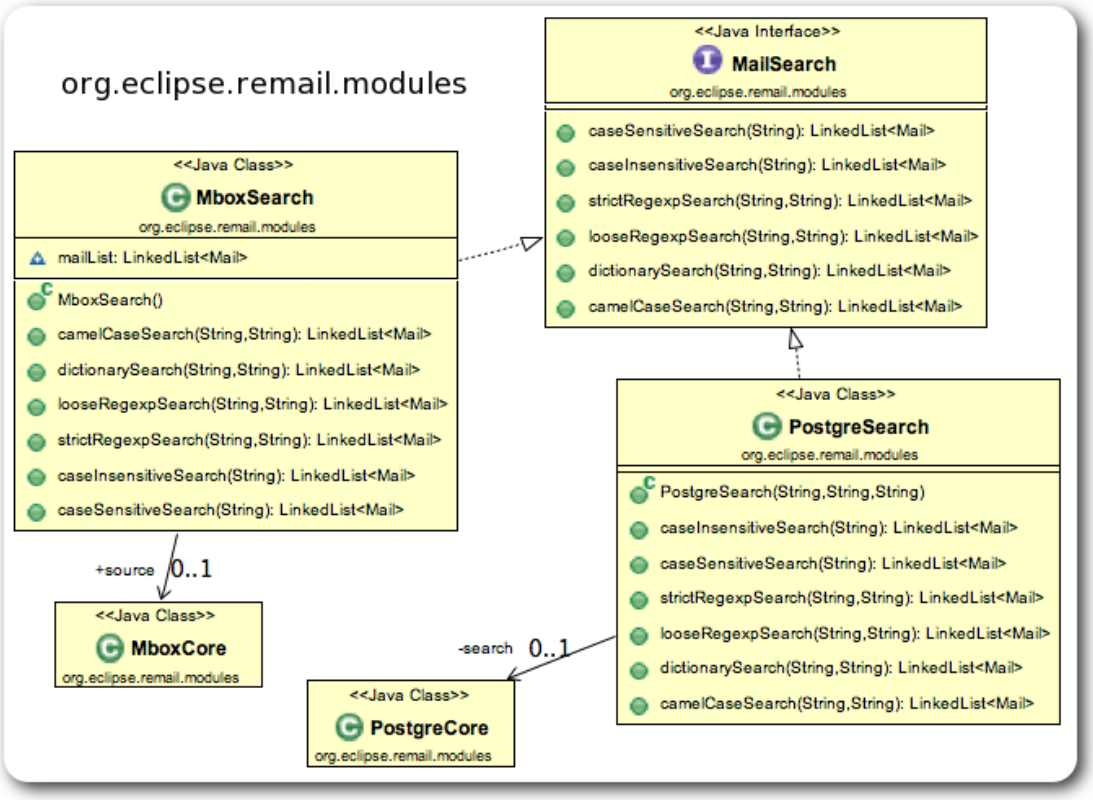


Figure 3.10: Interface based modularity of data-source implementations

The interface *MailSearch* and its implementations are all part of the *modules* package. This is a completely non-UI package. Other packages contain classes manipulating the user interface. Almost all of these classes need to have access to the indexed results data, so that they can process it and present it in Eclipse.

Thanks to the way Eclipse invokes classes defined in plugin.xml <*extension*> tag, relations between UI packages are minimized, with the exception of using some of the code in *org.eclipse.remail.util* package, which contains generally useful classes.

The class *Search* in the root *org.eclipse.remail* package performs the search by using one of implementations of the *MailView* interface, according to the current preferences settings. The class *Search* is the only class to directly call the *MailSearch* interface. It ensures execution of linking procedures between a single source code element (a class) and a mailing list of choice. The list of *Mail* instances is returned to be employed when necessary.

org.eclipse.remail.Mail is a data class that serves as a central representation of a single e-mail throughout all REmail. In addition to simply containing necessary attributes (subject, author, text...) it provides utility methods for merging lists of *Mail* objects etc. *Mail* also implements the interface *Comparable*, which allows collections of *Mail* objects to be sorted by the timestamp of an e-mail.

3.3.2 Result indexing

In addition to the *Mail* and *Search* classes, *SelectionProcessor* and *IndexSearch* are also parts of the root *org.eclipse.remail* package.

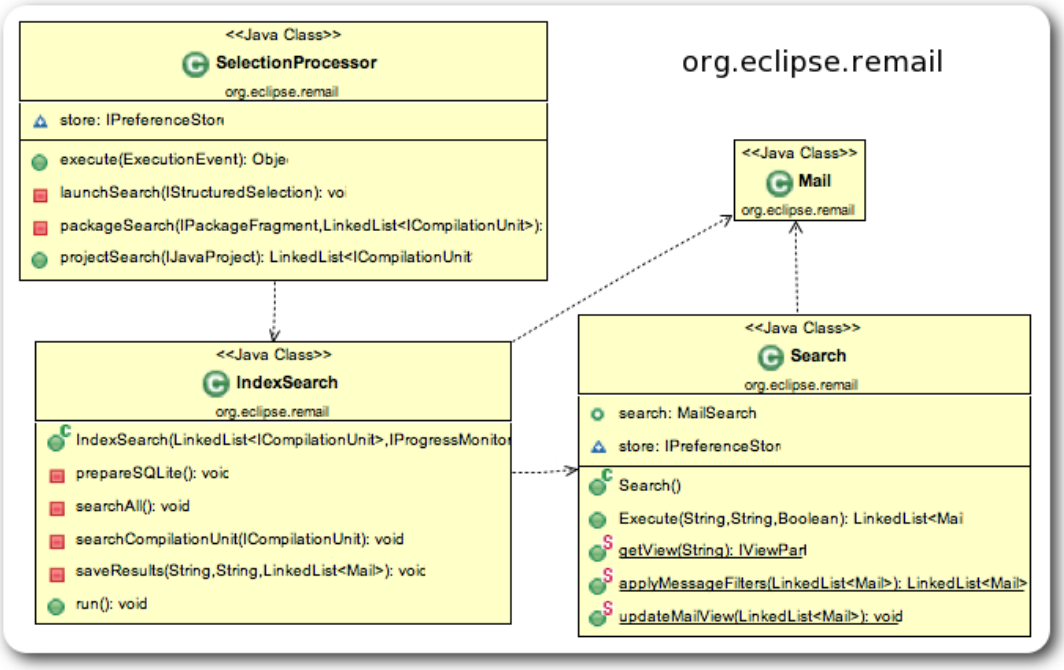


Figure 3.11: Processing of the indexed search

SelectionProcessor This class is listed inside *plugin.xml* as a handler for the *REmail search* command, which serves as an initiator of search and is shown in the context (right click) menu of the Package Explorer. Since user can select multiple classes, packages, or the project itself, the task of *SelectionProcessor* is to produce a list of the actual classes [11] to be submitted to the search. *SelectionProcessor* then submits the list to the *IndexSearch* to continue the work.

IndexSearch Based on the given list of classes to search, *IndexSearch* commits the linking and subsequently indexing process on all of them. It depends on the *remail.modules.Search* class. *IndexSearch* needs to manipulate UI to inform the user about the progress of the search, which is the reason why it has been included outside the non-UI modules package. Inside *IndexSearch*, we use the Eclipse status bar *progressMonitor* extension to work with the progress bar that notifies users about proceedings of the search.

As also stated in previous sections, the search results are indexed. Indexing is necessary for providing vital functionality to the plug-in - the Package Explorer decoration or the editor integration. Fast displaying of the linking results for previously searched classes is an example of indexing usage.

[11] List of *CompilationUnit* instances, which in Eclipse represent the *.java resources (or other source file types, depending on the Eclipse distribution)

Indexing is implemented using a database approach. SQLite provides a JDBC driver, which allows us to work with structured data using SQL language - in the same way as with most of the database management systems. Difference is that SQLite does not run as a server, and no actual connection is needed. Instead SQLite is basically a library, that lets us store data in a single file in a transparent way.

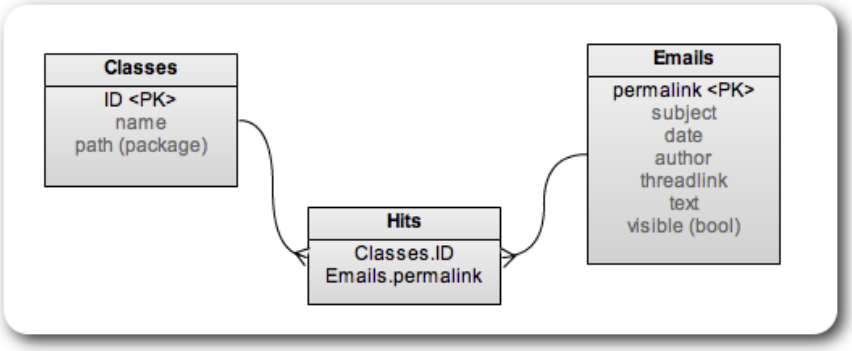


Figure 3.12: SQLite DB structure for search results indexing

To implement the indexing, we use three tables that employ the typical many to many (N:M) approach known from the relation databases theory [12]. As shown in the entity relationship diagram on figure 3.12, this is implemented using two data tables, (*classes* and *mails*), which use a third table (*hits*) to provide N:M relations.

3.3.3 Source formats

As explained previously, REmail's modular structure allows for any number of different searching techniques to be implemented. Currently two data sources and the related techniques are implemented for searching as shown in figure 3.10.

PostgreSQL

For the PostgreSQL search, class *PostgreSearch* implements the *MailSearch* interface. This class channels the results of the search from the *PostgreCore* class that is actually conducting the search. As we use JDBC driver to access PSQL database, *PostgreCore* is where actual SQL language is used.

```
ResultSet rs = stmt.executeQuery(
„SELECT * FROM mail WHERE rawcontent ~* '(\\s*)(“ + restOfPackage
+ „)?(\\.|\\\\|/|\\s)“ + packageLastPart + „(\\.|\\\\|/)“ + classname
+ „(\\.java|\\.class|\\s+)' order by timestamp“);
```

Methods of *PostgreCore* initiate SQL queries, and put results into the list of Mail objects to use. SQL queries contain regular expressions designed on the theoretical description of the lightweight linking methods [2]. An example of using SELECT statement to get results by "Strict regular expression" method is provided above. The parameters needs to be inputed from java variables and the whole statement must be escaped. We give an actual usage as in the code as an example, instead of plain SQL statement.

As we use SQL and database in the PSQL module, it should not be confused with SQLite DB usage for preserving results. SQLite serves as REmail's internal data storage tool, while PSQL serves as an source of original data. When MBox search is selected in the preferences, REmail does not maintain any actual database connection. Description of setting up the PSQL and the necessary structure of the data table are provided in next section.

MBox

Another implementation of the *MailSearch* interface provides MBox data source. Class structure of this implementation is the same as in the case of PSQL core (this can be seen in class diagram 3.10). *MboxSearch* prepares the input variables for the search, which is done using *MboxCore* class.

However, differently from the *PostgreCore*, *MboxCore* does not do the complete search. It merely provides for case insensitive method, which is basically just matching a name of the class in question. All the results of case insensitive search are always a superset of the results of any other method. *MboxSearch* uses Java's regular expressions to get a subset of the raw results using a chosen method. The reason of this is for obtaining better performances. For example, imagine that we search through mailing list of 25000 e-mails and the average number of results is around 30, using *CamelCase* method. Having to apply regular expression on all the e-mails takes a considerable amount of time, more then just searching for a single classname. Applying the regular expressions on the resulting 30 e-mails is orders of magnitude faster.

MboxCore accesses the location of the MBox file of the current mailing list from the plug-in's preferences storage. Its methods then pass through the file in search of the classname. If there is a match in the text of any scaned e-mail, it retrieves the headers and stores a *Mail* object in the list.

3.3.4 Views

Realizations of both views of REmail are placed in the *org.eclipse.remail.views* package. Both are completely UI-centric as they must present the search results.

MailView

The *E-mails* view is the main point of presenting the results of the linking process. The view is defined in *plugin.xml* and is implemented in *MailView* class. *MailView* extends Eclipse's abstract class *ViewPart* (mandatory for any view). Method *CreatePartControl* is also obligatory and serves as a point of initiating the UI parts of the view.

MailView uses a single control: - an instance of JFace *ContainerCheckedTreeViewer* that completely fills the view. This allows to create a combined table/tree element, the best solution for presenting search results of the REmail. In addition, in the beginning of every row, it also provides a checkbox, which we used to create the irrelevant e-mail removal feature.

MailView is a the largest class of REmail, as all the presenting of results is implemented in it. Four subclasses are necessary for that. Input of the *ContainerCheckedTreeViewer* is simply a *LinkedList* of *Mail* objects. *MailTreeContentProvider*, which implements interface *ITreeContentProvider* contains three methods telling the *ContainerCheckedTreeViewer* the structure of the given data. Notably, first the top level elements need to be defined. After that, if some of them have children, those need to be pointed out.

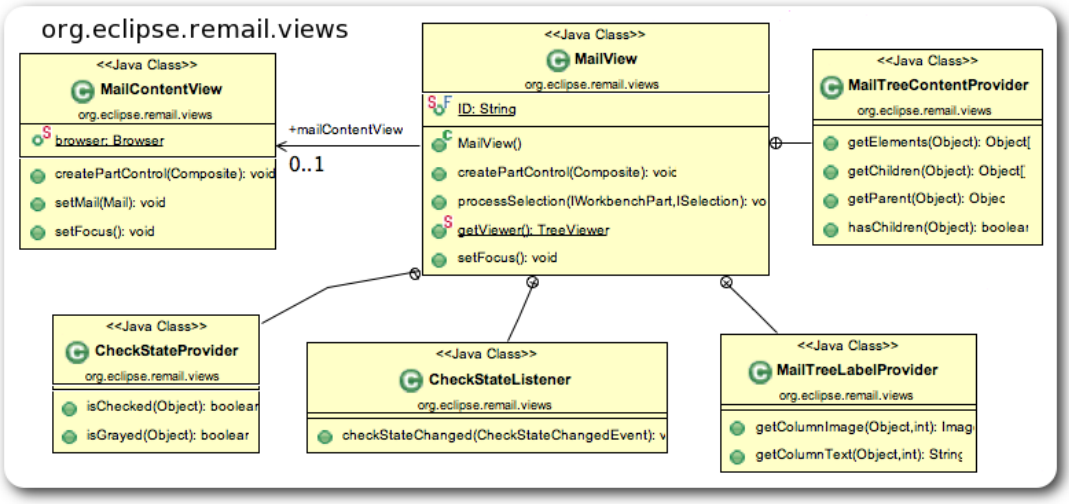


Figure 3.13: REmail's implementation of views

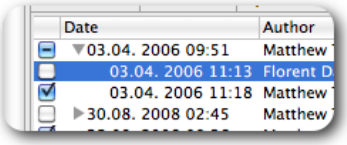


Figure 3.14: Looks of the current implementation of the MailView class

MailTreeLabelProvider is much simpler than *MailTreeContentProvider*. Given a single *Mail* object, it decides what text should appear in different columns of the viewer. That is made possible by implementing *ITableLabelProvider* interface.

The other two subclasses, *CheckStateProvider* and *CheckStateListener*, manage the check-box feature of the *ContainerCheckedTreeViewer*. The former class defines which e-mails are to be checked or otherwise while displaying results. The latter is a listener that is informed when user checks or unchecks any box. The indexed information is updated in the SQLite database.

MailView incorporates a listener that waits for changes of selection in the Eclipse workbench. If an instance of *ICompilationUnit* has been selected, a method of *MailView* will check whether any data had been indexed. If so *MailView* shall give this *ICompilationUnit* to a new instance of *SQLiteMailListConstructor* (in the *org.eclipse.remail.util* package) to produce a *LinkedList<Mail>* from the cache. After that, this list is put as an input.

MailContentView

MailContentView implements the *E-mail content* view. This view simply shows the contents of the e-mail chosen in the *E-mails* view. It is plugged into Eclipse the same way as previous view, by being defined in the *plugin.xml*. It also uses one widget that fills entire view. That is a Browser widget, which is a part of SWT. This widget can display any html formatted content. The e-mail to display in the view is set up using the *setMail* method, which takes

a *Mail* object as an argument.

The view displays the e-mail in an user friendly way, taking advantage of the stylesheet support. A class *ContentDecorator* in the *util* package has been created for that purpose.

3.3.5 Editor integration

As introduced in the evolution Section 3.2, REmail integrates itself into the general Eclipse source code editor, by giving possibility to show markers for lines that contain a name of class that has been searched for before.

The *org.eclipse.remail.editor* package contains the implementation. Package contains two classes of which *markerInitActionDelegate* is the implementation of marker extension. The other class, *MailHover* is an attempted implementation of a Javadoc hover extension, that currently is under construction (explained in 3.2.5).

The *markerInitActionDelegate* is registered in plugin.xml as an action class for pressing of the button in the main toolbar. It implements interface *IEditorActionDelegate*, which defines a *run* method to be ran on action activation. From this method we can obtain the text of the active editor. The document is searched for known (i.e. indexed) classnames. In case of a hit, a *Bookmark* marker is added to the Eclipse environment. After the scan of the document is complete, Eclipse platform shows all the markers at the proper line on the left-side of the editor automatically.

3.3.6 Preferences

Several preference pages had been included into the Eclipse general preferences. Extension *org.eclipse.ui.preferencePages* is used to insert the preference page. All of them are put into the single „REmail“ preference category.

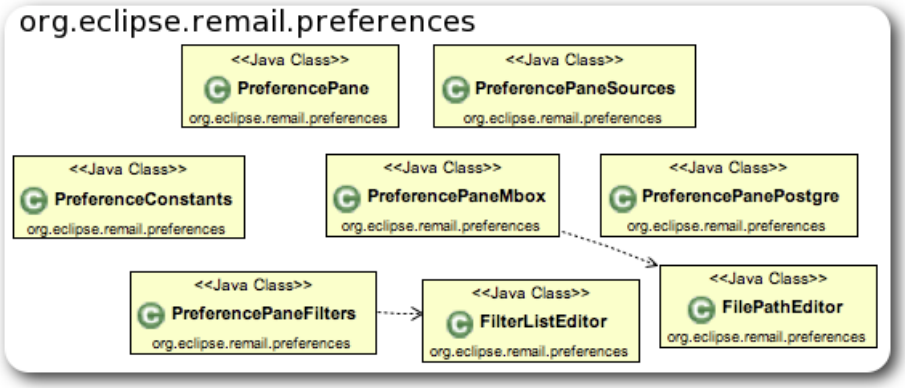


Figure 3.15: Classes in *org.eclipse.remail* package

As seen on class diagram 3.15, since all the preference pages and classes implementing them are defined separately in the *plugin.xml*, there is no need for any relation or dependency among the classes realizing different pages. All of them implement interface *FieldEditorPreferencePage*, which allowed us to use many common types of widgets to create preferences. Eclipse had already contained classes to create simple line editor, combo or radio button preferences.

These basic pre-programmed widgets were useful to create most of the preferences needed - with two exceptions:

- We had to create our own editor for entering message filters. We have extended the JFace *ListEditor* to provide this component. The *ListEditor* "Add" button was extended to show an input dialog, in which developer can type in the filter, and also actions on pressing "OK" and "Apply" buttons were changed to provide immediate change of the Package Explorer number decoration.
- We needed to have an editor to allow the user to select and store multiple files - for the management of MBox files storing different mailing lists. Similar editor for managing directories already exists, however we couldn't use or extend it. Therefore we have employed *ListEditor* and extended it to provide file choosing and storing functionality.

3.4 Using REmail

This section serves as an "User manual" with instruction on how to setup REmail, how to obtain mailing lists and how to use the plug-in itself.

3.4.1 Installation

At this time, there are two ways if puting REmail into action. First is by downloading the source code using the SVN repository at the project page on Google Code[x]. We suggest to use the Subclipse[x] plug-in. With Subclipse, go to *File->Import*, select *SVN->Checkout projects from SVN* and press *next*. On the next page choose to create a new repository and enter *http://r-email.googlecode.com/svn/trunk/*. When asked for login information, put *r-email-read-only*. On the last page, choose whether you'd want to put REmail into the workspace and after that you can checkout the project.

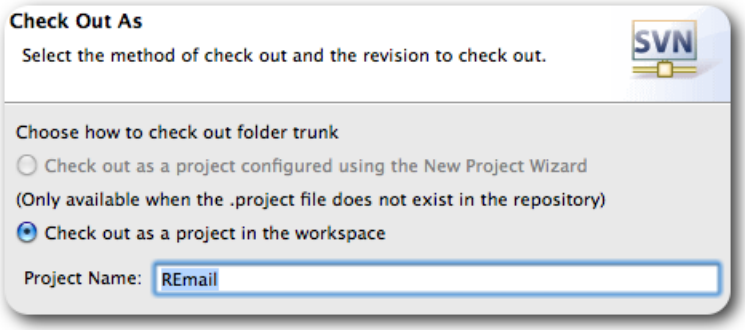


Figure 3.16: Importing REmail using Subclipse

After these initial steps you can see REmail as a project in a package explorer. Using this method, you need to press run to open second instance of Eclipse, in which the plug-in will be active. This way is good when you are interested in the REmail's implementation or you wish to contribute in it. If you wish to use REmail for its main purpose, consider using the other install method described later.

This first method also implies that you have Eclipse distribution for RCP/Plug-in developers installed.

The other possible installation consists in visiting the download section of REmail website[12] or its Google Code page[13]. There you can download *jar* file - a precompiled distribution of REmail. Copying the *jar* file into the *plugins* directory of your Eclipse, will cause the plug-in to be loaded on the next start of the platform.

3.4.2 Setting up

After having installed the plug-in, you must decide, which mailing list you desire as data source. We recommend the MBox file source, as we also made an automated tool available to download any open source mailing list into MBox file. If you prefer to use PSQL as a data source, you must create a table and upload data in format that will be described later.

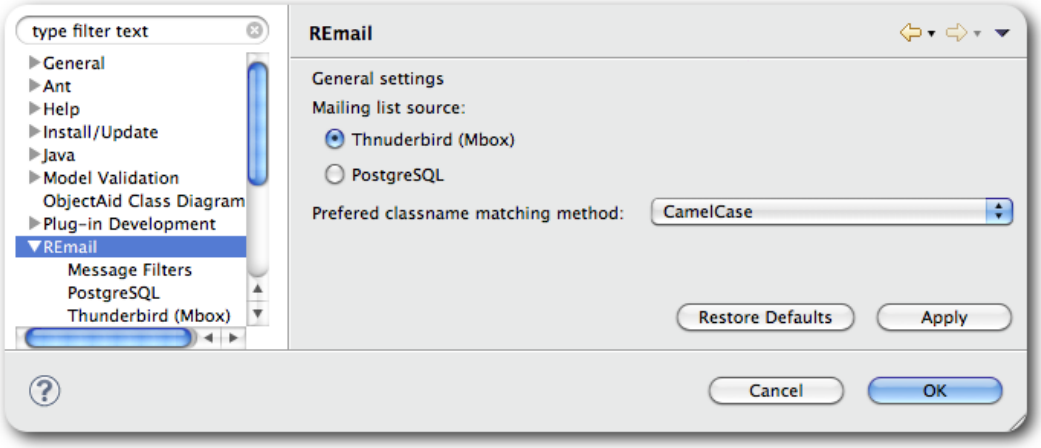


Figure 3.17: Main preference page

To set up REmail, open Eclipse's preferences, look for the REmail category and click on it. In the "REmail" preference page (Figure 3.17), you have to select a data source and lightweight e-mail-to-code matching method. For the beginner we suggest *CamelCase*, which provided best sets of results when the lightweight methods research was conducted[2], as well as during developing REmail.

After going through the REmail page, either PostgreSQL or MBox pages needs to be configured based on the choice of the mailing list source.

PSQL setup

Should you have decided to take advantage of PostgreSQL, you will have to set up a table with the given structure and fill it with the data of your mailing list. No automated tool to do that is currently available, however we are considering to create it.

In any PostgreSQL database, you can create the table using this the following SQL statement:

```
CREATE TABLE mail (
    threadpermalink character varying(255),
    permalink character varying(255),
```

[12] http://remail.inf.usi.ch/
[13] http://r-email.googlecode.com/

```
    author character varying(255),
    rawcontent text,
    subject character varying(255),
    „timestamp“ timestamp without time zone,
);
```

The names of most the attributes are self-explanatory: *Permalink* must be unique string identifying the e-mail, while *treadpermalink* should only be the same for all e-mails inside a single thread. If you plan to use dictionary search, you also need to create a table with single attribute named *word* and fill it with words you want to consider for the method. However, we encourage you to use *CamelCase* method for its simplicity and better performance.

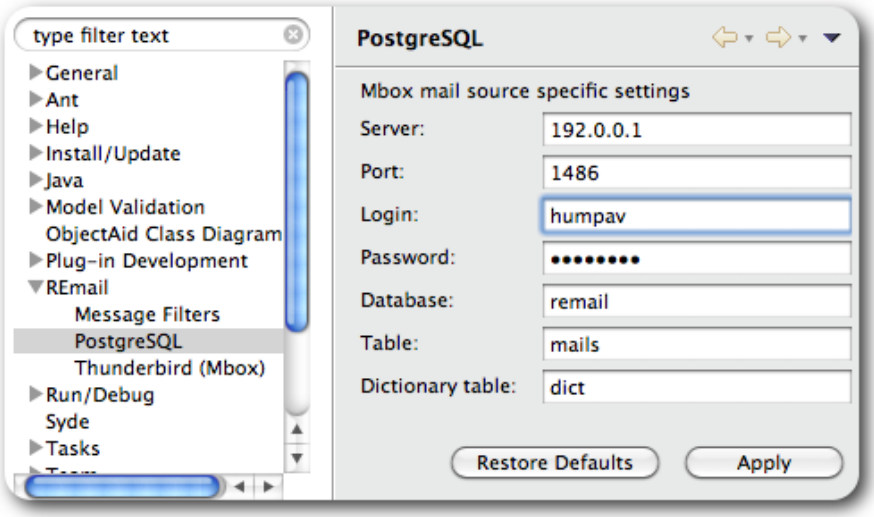


Figure 3.18: PostgreSQL preference page

To setup REmail to work with PSQL, fill in the information on "PostgreSQL" preference page (Figure 3.18). All fields are mandatory except for the "Dictionary table", which must be filled only if you plan to use the *Dictionary matching* linking method.

MBox setup

When using the MBox source of mailing list, first you need to get a MBox file. There is a number of possibilities for achieving this. For example, you can download it from official archives of the mailing list, or you can take advantage of the tool we provide.

Mailman Most of mailing lists of open source projects use Mailman as a list management system. If that is the case, you can often find archives on the webpage of the list. One drawback of this is, that they are usually split by month into separate files. However as they are already in MBox format, they can simply be joined together to provide a single MBox file. Currently we are considering making a tool to download and merge the files automatically.

Miler tool As have been mentioned before, we provide a tool to get the mailing list from the *markmail.org* site. This tool is based on Miler[3], and we have modified web crawler to

store the downloaded mailing list in MBox file, which can then be used by the REmail.

To use Miler tool, you must use following syntax:

```
java -jar miler.jar name_of_mailing_list /path/to/store/mbox/files
```

As first parameter, input the name of the mailing list as you see it in the browser's address bar while viewing the main page of the list on markmail.org (example in Figure 3.19).

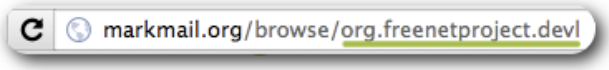


Figure 3.19: Getting a name of mailing list for Miler tool

The second parameter should ideally lead to a local folder of your e-mail client, so that you can use it to manipulate the list. REmail contains a feature to remove e-mails from the results, however for a bigger scale editing of the mail list, using an e-mail client might be preferable.

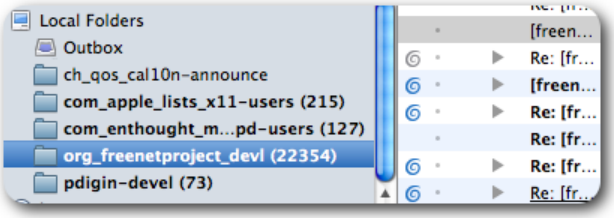


Figure 3.20: Example of manipulating mailing list in Thunderbird

Figure 3.20 shows how a mailing list, that was put inside Thunderbird's Local Folders, can be manipulated[14] as any other e-mail folder.

When a mailing list's MBox file has been obtained, path to it needs to be entered into the MBox preference page.

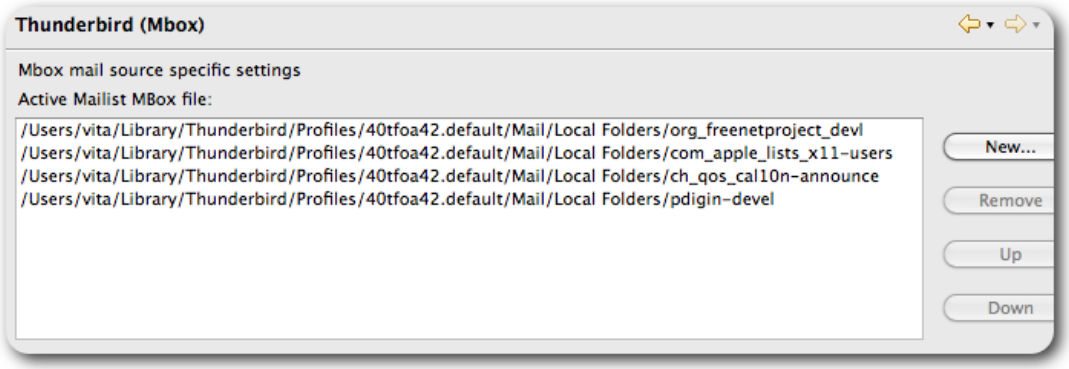


Figure 3.21: MBox location preference page

[14] If Thunderbird is used to remove unwanted e-mails, changes to the original MBox file are done after executing command "Compact" from the folder's context menu.

Such page contains a path editor, in which multiple mailing lists can be stored. A file on top of the list is the one that is active for the search. It means, that when switching to another project, you have to move appropriate list to the top. This is caused by implementational limitation of preferences and is likely to be changed.

3.4.3 Searching

After REmail has been set up, the next step is to proceed with the linking process on classes that one is interested in.

The package explorer is an entry point for all the search. You can select any combination of classes and packages or you can select the project itself. When invoking the context menu on this selection, you'll see "REmail search" command, that, if clicked, will initiate the search (see figure 3.22).

While searching, as the results are getting indexed, a progress bar (figure 3.5) at the bottom of the window will give feedback about the progress of the linking process. When the search is completed, the number of "hits" given by the selected linking method will appear next to classnames and packages in the Package Explorer.

3.4.4 Browsing e-mails

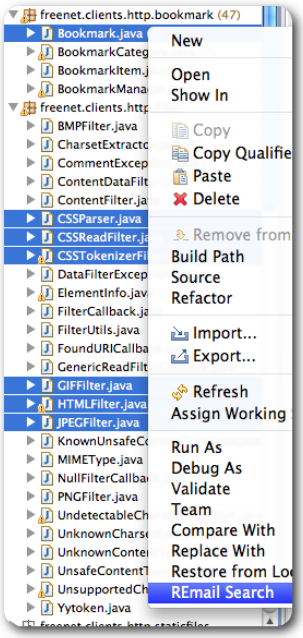


Figure 3.22: Starting search

To show results for any class, you simply select it in the Package explorer. The "E-mail" view will be automatically open the first time after installation of REmail. The standard location is at the bottom dock, but you can change the position.

You can browse the results in such a view, and if you find any single e-mail or an entire thread unwanted, you can uncheck it using the checkbox. In this way, you can express lack of interest any the e-mail or thread.

As you can see in the Figure 3.14, most of the relevant data are shown already in the E-mail view itself. However, if you stumble across an interesting result, you will most likely be interested in the message text itself.

That is the purpose of "E-mail content" view (fig. 3.23). Double-clicking any result in the "E-mail" view will display its full content there. Every message is displayed with header on top and with the classname of interest highlighted in red. Parts of the text from other messages in the thread are indented and printed in different colors.

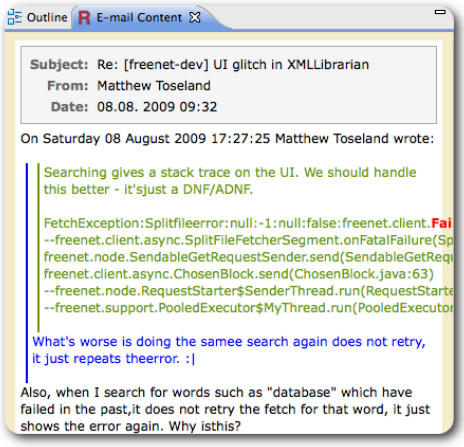


Figure 3.23: The "E-mail content' view'

3.4.5 Message Filtering

REmail is capable of filtering messages based on the content of Author and Subject headers. You can set up filters, that will apply on all of the results shown in E-mails view, by adding them on "Message Filters" preference page. Currently REmail looks for a presence of the string in a header to include or exclude the results from the list.

It is worth noting, that *all* the messages ale always cached when searching. Message simply merely determines their presence in the E-mail view and the counted hit numbers in Package explorer, without removing them from the DB. In this way different filters can be switched with no impact on the performances.

3.4.6 Editor Integration

REmail contributes a trigger button onto the main toolbar.

By pressing this button, you can toggle the bookmark markers in the active editor. As you can see in the figure 3.8, a marker will appear on every line, which contains a classname that has been submitted to the search. By moving mouse over a marker, a hover will show you the number of hits the class has with the current filters active. In case there are more class entities on the same line, just one marker will be shown, however the hover will include information about all of the classnames.

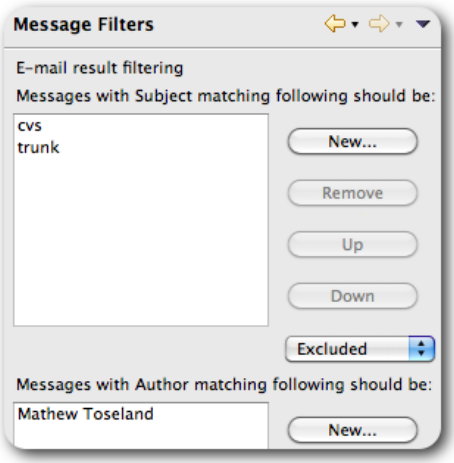


Figure 3.24: Filtering out messages

Chapter 4

Case Study

During the development of REmail, we used *Freenet*[1], the software system developed by the Freenet Project, to test the implementation and validate the functionalities.

The Freenet Project aims at creating a free software system that lets you anonymously share files, chat on forums, and browse and publish "freesites" (web sites accessible only through Freenet), without fear of censorship. Freenet is decentralized to make it less vulnerable to attack, and if used in "darknet" mode (where users only connect to their friends) is very difficult to detect.

Freenet is in active development since 1999, is written in Java, and has an active community of developers who also use e-mails for communicating. By the time of writing this thesis, Freenet consisted of over 850 classes and 32 packages, and the development mailing list archived more than 20,000 messages.

For the purpose of exploring Freenet with the assistance of REmail, we checked out the source code of Freenet using its *git* repository[2], and we imported it as a project into an Eclipse workspace.

4.1 Choosing a linking method

The most important part of working with REmail is the process of linking the chosen classes, or packages, to the e-mail, and then investigating the results. Therefore, before providing any extra features, we first implemented the different linking methods (as depicted in Section 2.2). We have implemented such methods gradually and we explored the results step by step.

First, we have experimented with *case insensitive* and *case sensitive* linking techniques. As we expected (see Section 2.2), the case insensitive method produced results with high recall and low precision, confirming the results reported by Bacchelli *et al.* [2]. On the other hand, the case sensitive method achieved a higher precision with almost no impact on the recall achieved by the previous method. For this reason, we removed the case insensitive method from the latest version of REmail, thus increasing the ease of use of the tool.

Classes that have a common dictionary word as their name (*e.g.,* the classes "Metadata" or "Global") are the most problematic for the case sensitive method. In Table 4.1, we summarize the numbers resulting by linking several classes of Freenet with the linking methods implemented. For the case sensitive match, we see that classes with common

[1] http://freenetproject.org/whatis.html
[2] Instructions at http://freenetproject.org/developer.html

words as names have significantly more links than those whose names are in CamelCasing. Such additional links are often unrelated to the chosen classes.

class	*case sensitive*	*strict r. exp.*	*loose r. exp.*	*CamelCase*
Metadata	335	26	26	26
Global	104	1	1	1
Yarrow	61	11	12	11
ConfigToadlet	54	6	8	54
ArchiveManager	43	35	35	43

Table 4.1: Numbers of results of linking different classes with various techniques

Strict and *loose* regular expression methods provide less links for single word classes, practically all of them correct. These methods, on the other hand, can be to be too strict for classes in CamelCasing: Since such classes are very often just mentioned by their names, without parts of package or file extension, they were not registered by the *Strict* and *loose* regular expression methods. Thus, important information can be lost. Figure 4.1 presents an example of an important e-mail not retrieved with such methods.

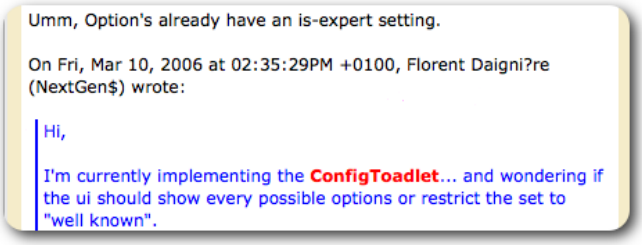


Figure 4.1: E-mail related to a class, retrievable *only* by the case sensitive method

The *CamelCase* technique deals with this issue in a straightforward way: As previously explained (Section 2.2), such technique offers the best trade-off between precision and recall, while maintaining reasonably high value. As depicted in Table 4.1, this method runs the *strict* regular expression search only on classes named by a single word, while it uses the *case sensitive* approach on class whose name is formed by more than one word. In this way, it combines best of both techniques.

By applying the *CamelCase* technique on Freenet, we see a great difference in package results, compared to the other methods. The package *client* serves as an example of this.

The *CamelCase* technique linked 291 e-mails to the *client*'s classes. *Client* contains 37 classes and amongst them only class *Metadata* is not in CamelCasing. Thus, for this class, the *CamelCase* applied the *strict regular expression* method, returning only 26 results. The case sensitive search, on the other hand, returns 335 results for this class, raising a number of unique e-mails linked to all of package classes to 534. All of new e-mails are linked only to Metadata and are mostly irrelevant.

Using REmail with Freenet confirmed the results established by Bacchelli *et al.* [2] for each method. We used the CamelCase technique for most of the development and testing time, as we found it to be the most useful for our needs.

4.2 Refining results to obtain relevant information

4.2.1 Applying filters

Even though the CamelCase method provides results with a reasonable precision and recall, we also usually obtained a considerable number of e-mails, that indeed referred to a class in question, but are irrelevant in the context of program comprehension. In the case of Freenet, the vast majority of such messages were automatically generated and sent by the version control system for detailing commits. These e-mails include listings of all classes that are part of each commit. Short in nature, the related commit message is hardly relevant to the listed classes.

With such e-mails a new phenomenon have appeared: Even the *strict regular expression* method returned such e-mails as relevant links, because classes were listed using their complete path inside project structure, which followed the package structure. Even though the link is formally correct, these e-mails are in fact irrelevant (an example of this is in Figure 4.2).

Figure 4.2: Related, however irrelevant, e-mail linked by *Strict* technique

After observing such behavior, we decided to implement a message filtering (see Section 3.4.5) for allowing the user to reject messages based on subject and author fields. Such feature immediately proved to be helpful: The e-mails posted to the mailing list of Freenet by the version control systems have a special subject, thus, by creating a filter, we were able to remove all the unnecessary e-mails: Messages posted to the Freenet development list by *cvs*, are marked by the specific string shown in Figure 4.3. Therefore, by setting up filtering using it, all these e-mails are removed from the results and neither they appear in the *E-mails* view nor they are counted in the number of "hits" in the Package Explorer.

Figure 4.3: Example of filtering e-mails posted by version control systems

4.2.2 Selective result removal

Even though the *cvs* messages were removed by filtering, we could see that other messages that should be removed are difficult to filter in this way. E.g. the Freenet translation messages in Figure 4.3. Therefore we decided to implement means of selective result removal. By toggling the checkbox next to each e-mail, we can express the lack of interest in it. This is done selectively, e-mail by e-mail. Because we could often evaluate the importance of an e-mail simply by its subject, removing irrelevant messages using checkbox is not a time consuming task. As expected, this feature together with filtering helped to refine a list of e-mails that are better suited for content examination.

4.2.3 E-mail readability

At first, the text of the e-mail under examination was displayed in the *E-mail content* view very simply. It was not always easy to find the section of e-mail referring to the class. For this reason, all the classnames are now highlighted in the text of e-mail. With this, it is easier to find and understand the context in which the class was mentioned, and whether it is of any use. In addition e-mails often contain the text of previous messages belonging to the same thread. For this reason we have modified the view to display text coming from different messages in color. This increased the readability of the e-mail.

Message filtering together with selective removal and possibilities of quickly deciding the relevance of e-mails in the text proved to be effective when we were examining classes of Freenet to test the REmail and its usefulness. After choosing a class of the interest, we were able to rid the results of irrelevant e-mails and find potentially useful information in a reasonable time.

4.3 Other

Linking the e-mails to the source code and retrieving information to help developer's program comprehension is the core of REmail's functionality. However as we were developing REmail, testing it and getting feedback, we found it necessary add more features and functionalities, which we have subsequently tested:

- We attempted to obtain source code understanding of several classes when working solely with the editor. By toggling the bookmark markers we could see the numbers of all the e-mails related to all the classes mentioned in the code. Classes with a greater number of linked e-mails were discussed more, thus were a good candidates for further examination, especially because the original class of interest was dependent on them.

- Similarly to the editor improvement, the presentation of number of links per each class and package in Package Explorer allowed us to quickly pass through packages and see the most discussed classes - thus the most important or problematic. However (mostly when not using CamelCase linking technique) we discovered that we needed to be careful, as certain classes had a greater number of "hits", but were mostly unrelated because of their names are dictionary words.

- At the beginning the retrieved e-mails were presented on a single list. While inspecting the search results, we found that most of the e-mails were containing bodies of other e-mails in the list. These messages were originally parts of the same thread. Therefore,

we have modified both source modules to obtain thread related information as well. This allowed us to create a threaded result presentation. When investigating threads, we were mostly examining only the most recent message as it usually contains parts of the other e-mails, providing a entire thread's discussion at one location. This generally speeded entire process of result inspection.

- Initially, the *Mail* view was implemented as a simple table that could be sorted by it's columns. Because of the combined table-tree viewer (that we had to use when we introduced threads) and the temporal nature of e-mail succession, all the e-mails are sorted only by time. This took away a possibility of sorting e-mails by the subject and author. While sorting by author was sometimes useful, sorting by time brought related e-mails together because similar development issues are discussed in the same period of time.

Chapter 5

Conclusions

5.1 Summary

We have created REmail, an Eclipse plugin that integrates e-mail communication in the IDE. We have been motivated by the growing need of improving how developers, especially when geographically spread, communicate. We have focused on e-mail communication: The most common mean of communication for distributed teams, and we believe is one of the best suited for integration with the source code.

REmail incorporates several lightweight techniques to link classes to e-mails discussing them. We have extended the GUI of Eclipse in a number of ways to allow developers to take advantage of these techniques seamlessly.

The features we have implemented can be a useful tool for developers trying to comprehend a new software system: A developer can choose any of the mailing lists present in the MarkMail service, use the tool we have created to import it, and use it with REmail. The plug-in is implemented in a modular way, currently allowing programmers to utilize two possible data sources: mbox e-mail storage file format and PostgreSQL DBMS.

REmail offers a configuration interface to allow programmers to properly set their preferences. The central points of our plugin are: the "Package Explorer", where developers select classes they are interested in and submit them for the linking process, the "E-mails view", where the results of linking are presented to the developer, and "E-mail content" view, which presents individual e-mails in a structured and enhanced way. REmail is also integrated in the source code editor, where programmers have access to the results of linking without the need to have the additional views open.

During the development, we have used the Freenet software system and its development mailing list as a case study. This gave us immediate feedback on the usability of REmail and allowed us to improve it in order to achieve the most out of the linking techniques. We also have many ideas for other improvements that we plan to implement as future work.

5.2 Future Improvements

REmail is still evolving: Even though its current features and capabilities make it an useful tool for any Java developer, we plan to improve its usability, internal structure, and more.

Mailing lists management

The ability to obtain any desired mailing list and make it available for REmail is essential. We plan to provide users with other modules to import mailing list from other sources.

- Currently, a user can obtain a mailing list from *MarkMail.org* free service using modified *Miler* tool. However, this method of acquiring e-mail data has a limitation: The web crawler that is part of *Miler* is dependent on the *MarkMail* website. During the development, the structure of *MarkMail.org* has changed, forcing us to update *Miler*. Also, there is no guarantee that *MarkMail* will continue providing its service in future.

  As introduced on page 29, most of the mailing lists are managed by MailMan, the *GNU Mailing List Manager*. This system usually offers access to e-mail archives on the subscription page of a list. We plan to create a tool that will download archived e-mails, merge them together into single mbox file, and make them available for REmail.

- We also consider adding a new data source module, giving more flexibility to the users when working with the mailing list. Particularly, a support for obtaining e-mails stored online via IMAP accounts can be implemented. This would remove the dependency on an external e-mail client to manipulate the content of the list.

Usability features

- We are considering adding a support for direct management of mailing lists in the Eclipse. We would like to allow user to view an entire mailing list, not just results of a search, and allow reading, editing and deleting e-mails in it. We want to remove the need of an external application for doing this, while still having a mbox storage file that can be adressed by other programs.

- Currently, REmail allows a developer to work with e-mails extensively, however passively. We plan to implement a support for sending e-mails directly from the IDE, also with automatic keywords generation, after selecting a class or a piece of source code that programmer is interested about.

- We plan to offer mail-to-class traceability. This will give developers an access to source code of classes that are being mentioned inside e-mails they are viewing in the moment.

- We intend to add a not-intrusive notifier that tracks new e-mails about classes in which a developer is interested.

- We tried to extend a standard hover window in the editor to display a preview of search results related to the hovered class. Initially, we have failed to find how to implement this feature. However, recently, an answer on a forum question we posted introduced a method we could employ.

- When using mbox search module, developers have to choose the mbox source file in the preferences. This is implemented using a list, with the active one on the top. Now we want to let the user to link a number of mailing lists to specific projects in workspace.

- The "MarkMail" tool has to be launched separately to download a mailing list. We want to create a preference page, where the user can launch that tool and any similar future tools, thus merging all the work related to setting up a mailing list in REmail.

Other

How we previously explained in 3.1, we have chosen Eclipse for implementing REmail. The idea of the e-mail integration, however, can be put in practice into virtually any other IDE, thus we considering creating a version of REmail for NetBeans.

Additionally: REmail can currently be used with any programming language that Eclipse supports. However, since search is always started on classes or packages, the linking methods that we use are currently Java and OOP centric. Therefore, we would like to extend the e-mail-to-code capabilities of REmail to deal with larger variety of source code entities.

Bibliography

[1] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28:970–983, 2002.

[2] A. Bacchelli, M. D'Ambros, M.Lanza, and R. Robbes. Benchmarking lightweight techniques to link e-mails and source code. *In Proceedings of WCRE 2009 (16th IEEE Working Conference on Reverse Engineering), pp. 205 - 214. IEEE CS Press, 2009.*, 2009.

[3] A. Bacchelli, M. Lanza, and M. D'Ambros. Miler - a tool infrastructure to analyze mailing lists. *In Proceedings of FAMOOSr 2009 (3rd International Workshop on FAMIX and Moose in Reengineering).*, 2009.

[4] A. Bacchelli, M. Lanza, and V. Humpa. Towards integrating e-mail communication in the ide. *In Proceedings of SUITE 2010 (2nd International Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation), IEEE CS Press, 2010.*, 2010.

[5] A. Bacchelli, M. Lanza, and R. Robbes. Linking e-mails and source code artifacts. *In Proceedings of ICSE 2010 (32nd International Conference on Software Engineering), to be published.*, 2010.

[6] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. *In Proceedings of ICSM 2003 (19th IEEE International Conference on Software Maintenance)*, pages 23–32, 2003.

[7] E. Gamma and K. Beck. *Contributing to Eclipse: Principles, Patterns, and Plug-Ins.* Addison-Wesley, 2003.

[8] A. J. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. *In Proceedings of ICSE 2007 (29th ACM/IEEE International Conference on Software Engineering)*, pages 344–353, 2007.

[9] T. D. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: a study of developer work habits. *In Proceedings of ICSE 2006 (28th ACM International Conference on Software Engineering)*, pages 492–501, 2006.

[10] G.C. Murphy, D. Notkin, and K.J. Sullivan. Software reflexion models: Bridging the gap between design and implementation. *IEEE Transactions on Software Engineering*, 27:364–380, 2001.

[11] Web page: Network Working Group. Internet message format. http://tools.ietf.org/html/rfc5322, October 2008.

[12] Web page: Tom Jewett. Design pattern: many-to-many. http://www.tomjewett.com/dbdesign/dbdesign.php?page=manymany.php.

[13] Eclipse: Building Commercial-Quality Plug-Ins. *E. Clayberg and D. Rubel.* Addison-Wesley, 2 edition, 2006.

[14] E. Shihab, Z. M. Jiang, and A. E. Hassan. Studying the use of developer irc meetings in open source projects. *In Proceedings of ICSM 2009 (25th IEEE International Conference on Software Maintenance)*, pages 147–156, 2009.