REmail–Blending Talk and Work in Eclipse

Alberto Bacchelli *and* Lorenzo Baracchi *and* Michele Lanza

REVEAL @ Faculty of Informatics - University of Lugano, Switzerland

Abstract. Integrated Development Environments (IDEs) are isolated from those communication means that programmers use daily to interact with each other—as if talk and work were unrelated in software development.
We present REmail, an Eclipse plugin, that aims at filling this gap. REmail integrates email communication in the IDE, allowing developers to quickly and easily retrieve emails related to the class at hand, perform customized searches, or watch trends in discussions about code. REmail also allows developers to *produce* information, *e.g.,* by rating the value of any message or by sending contextual emails.

1 Introduction

In software development, teamwork has become the norm rather than an exception [7]. A team of developers working on the same project must deal with *coordination* and *collaboration* issues, such as effectively sharing the design rationale behind an implementation (*i.e.,* the most common information need for a developer [4]).

To coordinate and collaborate, collocated developers favor face-to-face meetings [4]– a communication means that causes frequent disruption of developers's attention (workers are interrupted every 7 minutes [6]), retains knowledge by few developers, cannot be archived effectively, and is unsuitable to globally distributed development projects.

Developers, thus, replace face-to-face meetings with electronic communication. Viable options are instant messaging, wikis, forums, *etc.*, but the decisive role is played by emails. Indeed, "Mailing lists are the bread and butter of project communications" [3]. Emails evade time zones, do not disrupt the flow of developers, and can be used to talk about issues ranging from mere bug fixing to high-level design. Mailing lists archive messages, thus offering a historical perspective, and broadcast discussions, announcement, and decisions to all the participants, thus maintaining team coordination.

Despite the importance of communication in software development, modern IDEs are isolated from this point of view. Even though developers spend most of their working time in IDEs (to write, read, debug, and understand code, and design new features) [5], they must interrupt their flow, switch the context, and use external programs (sometimes even a web browser) to manage their communication, particularly emails.

We present REmail, a plugin for Eclipse, which fills this gap. REmail was born as a lightweight recommendation system for emails to help developers finding discussions related to any class within Eclipse. It proved to be useful to improve developers' program comprehension and effectiveness [2]. Now REmail has grown and integrates email communication in the IDE: It includes features such as email writing, email data visualization, social rating, continuous updating, and fast text searching.

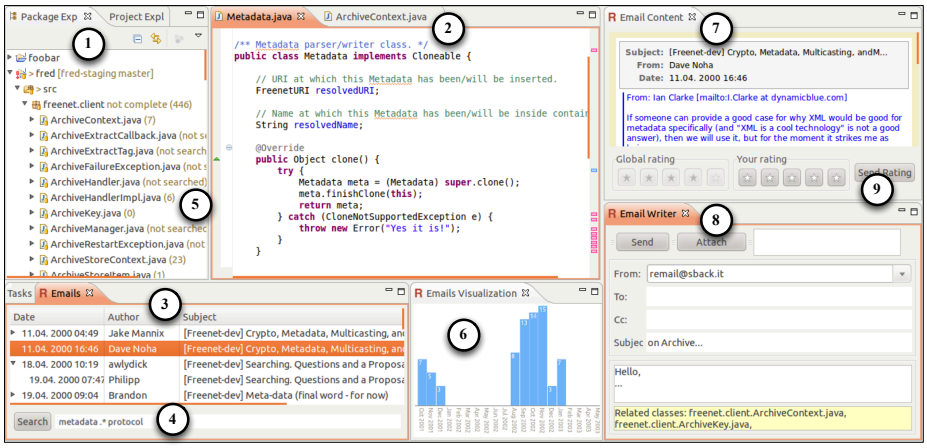

Fig. 1. The REmail Eclipse Plugin

2 REmail: User Perspective

Figure 1 shows REmail from the users' perspective. They can click on any entity in the augmented package explorer (1) or open/change a class in the editor (2), see related emails in the *Emails* panel (3), organized in threads and sorted by date. With the search box (4), they can further refine their search using keyword and/or regular expressions.

Users can perceive the relevance of classes from the point of view of mailing list discussions by means of two complementary perspective: In the package explorer, each entity is decorated with a number that shows the *cumulative* count of related emails (5), while in the *Emails visualization* panel (6), one can see how the related emails are distributed in time. This latter view shows the emails related to the chosen class as a bar chart: The x-axis is a discrete timeline, split in bins of equal duration, and the height is the number of emails exchanged during each period of time. This graph allows developers to see *trends* in discussions related to the chosen class. When users find a significant period and click on a bar, the *Emails* panel (3) shows only those emails. Once an email is clicked in the *Emails* panel, the *content* panel (7) shows its content with colors to denote quotation levels and emphasize the related class. REmail also enriches the code editor: When classes are mentioned in the source code, side markers give information about related emails.

With REmail, users can not only *consume* email information, but also *produce* it. By selecting any number of entities in the package explorer and/or a snippet in the code editor and clicking a button a new email is automatically prepared, whose body includes the chosen information. The *Email Writer* panel (8) shows the intermediate result. Users can then complete the message by hand. The code snippets and fully qualified names of entities are automatically included in the email body. Certain emails contain valuable discussions. Users can rate the importance of emails in the *Email* panel (9), this rating will be shared and averaged with the other users, thus creating a *social* rating.

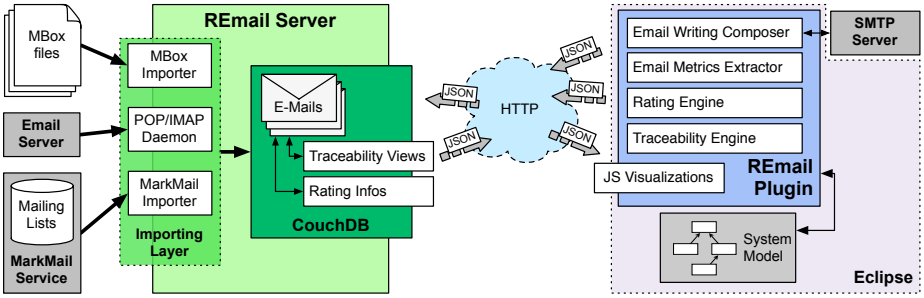

Fig. 2. The REmail's Architecture

3 REmail: Design Perspective

Figure 2 shows the architecture of REmail. Initially we devised a stand-alone solution, in which REmail was composed of the Eclipse plugin and simply used emails stored in the users' email client [2]. This approach had disadvantages: For example, emails had to be files in MBox format and could not be modified to avoid concurrency issues with the email client; developers working on the same project could not share information such as rating; any computation (*e.g.,* finding the links between classes and emails) is done client-side and must be replicated by each client. Now REmail is made of a server (left-hand side of Figure 2) and an Eclipse plugin (right-hand side).

The REmail server, to be installed by any organization willing to use REmail, relies on the document-oriented database CouchDB [1], which provides a RESTful JSON API accessible through HTTP requests. Being emails stored in a document-oriented database, we are able to change the meta-model without having to migrate data to a new schema, and we can use *MapReduce*[2] functions to parallelize tasks among cores or clusters of computers. To populate CouchDB, we devised an *Importing Layer* that handles different sources (*e.g.,* MBox archives, MarkMail[3]), extracts the selected emails, and instantiates them as documents in CouchDB. Moreover, we created a POP/IMAP daemon to handle newly received messages, thus keeping email data updated.

Any component of the REmail plugin that needs email data directly queries the CouchDB server through HTTP requests by sending and receiving JSON objects. As an example we see how the *Traceability Engine* (TE) works in practice. When users open a class from the *package explorer* or change tab in the main editor, REmail automatically triggers the TE. Since the TE retrieves the links between classes and emails by using lightweight text-matching techniques [1], it does not require additional information other than the fully qualified class name. The very first time users request emails for a class, REmail plugin generates a new specialized CouchDB "view" that implements the linking procedure, and permanently add it to the database. From that

[1] `http://couchdb.apache.org/`
[2] `http://labs.google.com/papers/mapreduce.html`
[3] `http://markmail.org`

moment on, every time emails for the same class are requested, the REmail plugin will query the view to obtain the emails. The first time the view is run, CouchDB applies it to all the emails to find the appropriate documents. The results will be stored and subsequent requests will be served almost in real-time. Moreover, since the view results are stored in a B-tree, when new emails are stored, the view will be updated accordingly in logarithmic time. The rating engine is based on CouchDB update and view functions.

The visualization panel (Point 6 in Figure 1) is based on HTML and Javascript. By using the SWT Browser, one can include HTML pages and interactive javascript within Eclipse and use java-to-javascript and javascript-to-java callbacks to make the view interacting with the environment. In this way, the same visualization can be accessed through a web browser outside of Eclipse (we can imagine a manager using this feature), all the effective javascript visualization libraries that are available can be used, and the IDE is more responsive than when using Eclipse visualizations.

4 Conclusion

We presented the key features of REmail (online at `http://code.google.com/p/r-email/`), an Eclipse plugin to integrate email communication in the IDE. Users can not only retrieve the emails discussing a certain class, but also write new messages, rate their importance, visualize email trends, or search for keywords. We plan extend REmail by (1) offering email-to-classes traceability (*i.e.,* go from emails to classes), and (2) adding a not-intrusive notifier to track new emails on selected classes.

Acknowledgements. Bacchelli is supported by the Swiss National Science foundation through the project "SOSYA" (SNF Project No. 132175).

References

1. Bacchelli, A., D'Ambros, M., Lanza, M., Robbes, R.: 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)
2. Bacchelli, A., Lanza, M., Humpa, V.: RTFM (Read The Factual Mails) –augmenting program comprehension with remail. In: Proceedings of CSMR 2011 (15th IEEE European Conference on Software Maintenance and Reengineering). pp. 15–24 (2011)
3. Fogel, K.: Producing Open Source Software. O'Reilly Media, first edn. (2005)
4. Ko, A.J., DeLine, R., Venolia, G.: Information needs in collocated software development teams. In: Proceedings of ICSE 2007 (29th ACM/IEEE International Conference on Software Engineering). pp. 344–353. IEEE Computer Society (2007)
5. LaToza, T.D., Venolia, G., DeLine, R.: Maintaining mental models: a study of developer work habits. In: Proceedings of ICSE 2006 (28th ACM International Conference on Software Engineering). pp. 492–501. ACM (2006)
6. Mark, G., Gonzalez, V.M., Harris, J.: No task left behind?: examining the nature of fragmented work. In: Proceedings of CHI 2005 (SIGCHI Conference on Human Factors in Computing Systems). pp. 321–330. ACM (2005)
7. Sarma, A., Bortis, G., van der Hoek, A.: Towards supporting awareness of indirect conflicts across software configuration management workspaces. In: Proceedings of ASE 2007 (22nd IEEE/ACM International Conference on Automated Software Engineering). pp. 94–103. IEEE CS Press (2007)