Miler: A Toolset for Exploring Email Data

Alberto Bacchelli, Michele Lanza, Marco D'Ambros
REVEAL @ Faculty of Informatics
University of Lugano
{alberto.bacchelli,michele.lanza,marco.dambros}@usi.ch

ABSTRACT

Source code is the target and final outcome of software development. By focusing our research and analysis on source code only, we risk forgetting that software is the product of *human* efforts, where *communication* plays a pivotal role. One of the most used communications means are emails, which have become vital for any distributed development project. Analyzing email archives is non-trivial, due to the noisy and unstructured nature of emails, the vast amounts of information, the unstandardized storage systems, and the gap with development tools.

We present *Miler*, a toolset that allows the exploration of this form of communication, in the context of software maintenance and evolution. With Miler we can retrieve data from mailing list repositories in different formats, model emails as first-class entities, and transparently store them in databases. Miler offers tools and support for navigating the content, manually labelling emails with discussed source code entities, automatically linking emails to source code, measuring code entities' popularity in mailing lists, exposing structured content in the unstructured content, and integrating email communication in an IDE.

Categories and Subject Descriptors

D.2.7 [**Distribution, Maintenance, and Enhancement**]: Enhancement

General Terms

Data Analysis, Human Factors, Tool Demonstration

Keywords

Toolset, Unstructured Data, Email Communication

1. INTRODUCTION

Teamwork has become the norm, rather than the exception, in software development [16]. Indeed, software is the product of *human* effort, which involves various activities other than writing source code. The actual code is the result of activities, such as information seeking, reflection, domain analysis, or design [11].

ICSE'11, May 21–28, 2011, Waikiki, Honolulu, HI, USA
ACM 978-1-4503-0445-0/11/05

When developers work in teams, these activities—which revolve around the production of source code—are pervaded by *communication*. For example, by studying how programmers spend their time at work, Perry *et al.* discovered that only 40% of programmers' working time is devoted to the assigned tasks, *i.e.,* coding [13], while most of their time is spent in communication [14]. Communication is how developers form and exchange their entire *knowledge* about the software systems they are developing [11, 12]. For these reasons, the investigation of communication and social aspects of software engineering is becoming an increasingly important field of research.

Developers' communication takes place either in face-to-face meetings or in electronic form, using means such as instant messaging, wikis, forums or emails. We focus on email communication, which is effectively used in both co-located [12] and distributed development teams and, according to Fogel, the "bread and butter" of communications in open source software (OSS) projects [10]. To our knowledge, there are no widely accepted methodologies, well-detailed and generalizable approaches, or comprehensive tools to conduct analyses and research on email data pertaining to software systems. While researchers enumerated the risks of using "off-the-shelf" techniques for processing mailing list data and showed how noise affects email content [8], most research dealing with emails employs *ad hoc* techniques tailored to specific research questions. Not only are these techniques scarcely described in the papers, but also their implementations are not publicly available, since they often consist of throwaway scripts, which do not offer the wider breadth and longer vision of full-fledged tools.

Our goal is to devise an approach for exploring email data pertaining to software projects, to investigate different facets of software evolution and development. We present the *Miler* toolset, which allows us to: (1) import mailing list archives from different formats; (2) model emails as first-class entities of a system, according to an extensible meta-model we devised [4]; (3) manually interact with emails and, for example, label them to create benchmarks for assessing the accuracy of mining methods [3]; (4) automatically find the traceability links between email and code entities, using various linking techniques [3, 7]; (5) automatically recognize emails and lines containing source code [2]; (6) export manual benchmarks and code metrics computed from mailing lists, such as "popularity" [1]; and (7) integrate email communication in the IDE [5, 6].

2. A WALK THROUGH MILER

Miler is a toolset to import, process, store, and analyze both email and source code data. Figure 1 presents the architecture: It shows the data sources from which Miler retrieves emails, the external components, the importers, the kernel, and the processing and analysis tools. Miler offers the following features:

Importing email data: Different software systems use different

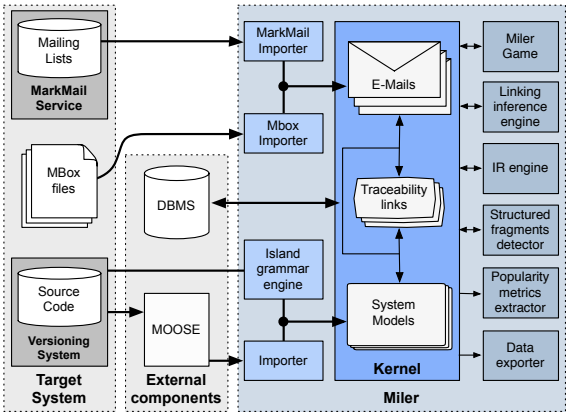

Figure 1: The architecture of Miler

applications to manage mailing lists, without offering a consistent data access. Also, a system can change mailing list application in its lifetime, i.e., it might be necessary to write multiple email data importers per software system. We tackled this issue by using MarkMail[1], an online service for searching among more than 8,000 up-to-date mailing lists. Our importer crawls the MarkMail website, extracts the selected emails, and instantiates them as objects in the Miler kernel. We also devised an importer for the MBox format.

Importing source code: Miler is implemented in SMALLTALK and handles software systems written in many programming languages through specialized importers. For JAVA systems, we use inFusion[2], for parsing the source code, and Moose [9], for extracting code metrics. The models created by inFusion and enriched by Moose are imported in the Miler kernel as *System Models*. We implemented an *island grammar engine* to import systems written in other languages. By using the concept of *island parsing* [17] and the SMALLTALK parser generator *PetitParser* [15], our engine parses ACTIONSCRIPT, PHP, and C systems, and consistently stores their models in the Miler kernel.

Interacting with emails: Using the SMALLTALK web framework *Seaside*, we implemented the Miler Game (MG), a web tool to interact with emails stored in the Miler kernel. Figure 2 shows MG main interface.

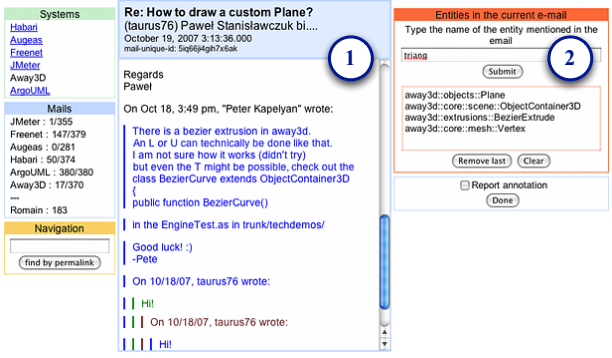

Figure 2: The Miler Game

MG has a modular structure organized in *panels*. Point 1 marks the main panel, which displays the selected email, with metadata on top and the whole body colored according to quotation levels to enhance readability. Within the MG, one can plug any number of panels to interact with the stored systems and emails. For example, on the left of the main panel, we see: the *navigation panel*, to read a specific email, given its id; the *system panel*, to switch to another system; the *mails panel*, with statistics on emails.

Linking code and emails: Even though development emails often discuss about source code artifacts, establishing actual links to the referenced entities is "left as an exercise to the reader". Moreover, the links are one way, there is no visible link from source code to emails. We implemented a *linking inference engine* to automatically infer these traceability links and persist them in the Miler kernel. The engine can use either lightweight text-matching techniques we devised [3], or the information retrieval (IR) techniques implemented in our *IR engine* (*i.e.,* LSI and tf-idf). With different case studies, we studied whether one of the techniques better handles this linking task [7]. To perform such a comparison, we needed an *oracle*, *i.e.,* a dataset in which emails are already linked to the correct code artifacts. Since it did not exist, we created it manually by reading thousands of emails. This task was completed with a specialized panel implemented for MG (Point 2, Figure 2). It allows the reader to annotate the email with the chosen links and supports the linking process in the following way: When users find a link, they can type the first letters of the artifact name in an autocompletion field; this triggers a menu that shows each entity, whose name includes the letters typed, colored following a special convention to display its time relation with the email. Users click on the right artifact to persist the link (this also avoids typos).

Detecting structured fragments: Development emails often contain structured content, *e.g.,* stack traces, patches, or code snippets. Separating structured fragments from natural language in email messages brings several benefits, such as better characterization of mailing list usage, improved authors' behavior analysis, or reconstruction of alternative system models [2]. Miler offers a *Structured fragments detector* tool, which implements a lightweight technique we devised for classifying emails and lines containing structured data. We created a benchmark to evaluate the effectiveness of a number of candidate techniques for detecting structured fragments in email content. This task was completed with a specialized panel for MG: It allows the reader to label email parts as structured, simply by selecting them with the mouse and using a keyboard shortcut.

Computing metrics and exporting data: From emails modeled in the Miler kernel, we can extract a number of metrics, ranging from simple measures (*e.g.,* number of email authors, or the lines of text in the contents), up to more complex ones (*e.g.,* number of emails with code). The Miler toolset offers a *Popularity metrics extractor*, which—by combining link data, system model, and email model—extracts various metrics to seize the "popularity" of code artifacts in mailing lists discussions [1]. Extracted metrics can be exported with the *Data exporter* component, into XML and CSV files. We used it to export the benchmarks manually created for testing our email-to-code linking techniques and structured fragments detection.

3. REMAIL IN A NUTSHELL

Remail is an Eclipse plugin to integrate emails in the IDE and allow developers to interact with them while programming. It distills the Miler toolset for software development and comprehension [6].

Figure 3 shows Remail at work. We can click on any entity in the augmented *package explorer* (Point 1) to see all the emails related to it in the *e-mails* panel, organized in threads and sorted by date (Point 2). Once an email is selected, we can read it in the *e-mail content* panel (Point 3), where colors denote quotation levels and emphasize the related entity. In the *package explorer*, numbers close to entities are one "popularity" metric—total related emails. Remail

[1] http://markmail.org
[2] http://www.intooitus.com/inFusion.html

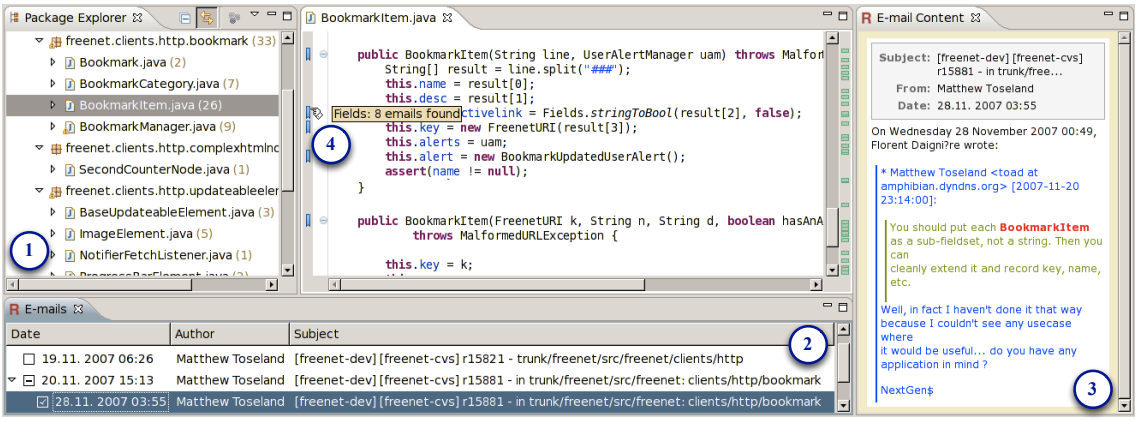

Figure 3: The Remail plugin for the Eclipse IDE

also enriches the *code editor*: When classes are mentioned in the source code, side *markers* give information about related emails. For example (Point 4), the mouse cursor hovers a marker in a line of code that uses the class *Fields*, and Remail informs us about the existence of 8 emails concerning this entity.

Figure 4 shows Remail's architecture: Remail can use email data stored in the database created by Miler, or in MBox files via an embedded importer; it embeds the *linking inference engine* and *popularity metrics extractor* to provide all their functionalities; and it takes advantage of the Eclipse IDE to access the system model.

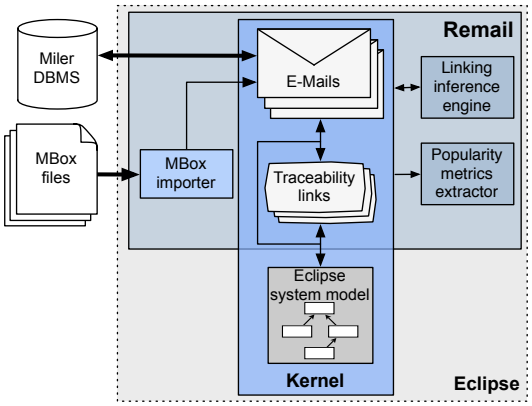

Figure 4: The architecture of Remail

4. CONCLUSION

We created Miler as a consistent, flexible, and extensible toolset for exploring email data. To our knowledge, Miler offers the most comprehensive and extensible solution for dealing with emails pertaining to software development. A video of our toolset is available at: `http://www.youtube.com/watch?v=MspFmsA1p_A`

Acknowledgements: The Swiss National Science foundation's support for the project "SOSYA" (SNF Project No. 132175) and the European Smalltalk User Group (`www.esug.org`).

5. REFERENCES

[1] A. Bacchelli, M. D'Ambros, and M. Lanza. Are popular classes more defect prone? In *Proc. of FASE 2010 (13th Int'l Conf. on Fundamental Approaches to Software Engineering)*, pages 59–73, 2010.

[2] A. Bacchelli, M. D'Ambros, and M. Lanza. Extracting source code from e-mails. In *Proc. of ICPC 2010 (18th IEEE Int'l Conf. on Program Comprehension)*, pages 24–33, 2010.

[3] A. Bacchelli, M. D'Ambros, M. Lanza, and R. Robbes. Benchmarking lightweight techniques to link e-mails and source code. In *Proc. of WCRE 2009*, pages 205–214. IEEE CS Press, 2009.

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

[5] A. Bacchelli, M. Lanza, and V. Humpa. Towards integrating e-mail communication in the IDE. In *Proc. of SUITE 2010 (2nd Int'l Workshop on Search-driven Development)*, pages 1–4, 2010.

[6] A. Bacchelli, M. Lanza, and V. Humpa. Rtfm (read the factual mails) - augmenting program comprehension with remail. In *Proc. of CSMR 2011 (15th IEEE European Conf. on Software Maintenance and Reengineering)*, page to be published, 2011.

[7] A. Bacchelli, M. Lanza, and R. Robbes. Linking e-mails and source code artifacts. In *Proc. of ICSE 2010 (32nd Int'l Conf. on Software Engineering)*, pages 375–384. ACM, 2010.

[8] N. Bettenburg, E. Shihab, and A. Hassan. An empirical study on the risks of using off-the-shelf techniques for processing mailing list data. In *Proc. of ICSM 2009 (25th Int'l Conf. on Software Maintenance)*, pages 539 –542. IEEE CS, 2009.

[9] S. Ducasse, T. Gîrba, M. Lanza, and S. Demeyer. Moose: a collaborative and extensible reengineering environment. In *Tools for Software Maintenance and Reengineering*, RCOST / Software Technology Series, pages 55–71. Franco Angeli, 2005.

[10] K. Fogel. *Producing Open Source Software*. O'Reilly, 2005.

[11] A. J. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *Proc. of ICSE 2007*, pages 344–353. ACM, 2007.

[12] T. D. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: a study of developer work habits. In *Proc. of ICSE 2006*, pages 492–501. ACM, 2006.

[13] D. E. Perry, N. Staudenmayer, and L. G. Votta. People, organizations, and process improvement. *IEEE Software*, 11:36–45, July 1994.

[14] D. E. Perry, N. Staudenmayer, and L. G. Votta. Understanding and improving time usage in software development. In *Process-Centered Environments*. John Wiley and Sons, 1996.

[15] L. Renggli, S. Ducasse, Gîrba, and O. Nierstrasz. Practical dynamic grammars for dynamic languages. In *Proc. of DYLA 2010 (4th Workshop on Dynamic Languages and Applications)*, 2010.

[16] A. Sarma, G. Bortis, and A. van der Hoek. Towards supporting awareness of indirect conflicts across software configuration management workspaces. In *Proc. of ASE 2007 (22nd Int'l Conf. on Automated Software Engineering)*, pages 94–103. ACM, 2007.

[17] O. Stock, R. Falcone, and P. Insinnamo. Island parsing and bidirectional charts. In *Proc. of the 12th Conf. on Computational Linguistics*, pages 636–641, 1988.