Miler – A Tool Infrastructure to Analyze Mailing Lists

Alberto Bacchelli, Michele Lanza, Marco D'Ambros
REVEAL @ Faculty of Informatics - University of Lugano, Switzerland

***Abstract*—The information that can be used to analyze software systems is not limited to the raw source code, but also to any other artifact produced during its evolution. In our recent work we have focused on how archives of e-mails that concern a system can be exploited to enhance program comprehension.**

In this paper we present Miler, a tool we have built for that purpose. We describe its architecture and infrastructure, and the FAMIX meta-model extension we have devised to model mailing list archives.

I. Introduction

In software systems, not only the source code, but any other artifact revolving around them (requirements, design documents, user manuals, bug reports, *etc.*) concurs to define their shapes. Such artifacts add information either by describing a specific piece of the source code or generally introducing concepts or necessities.

E-mail archives are widely employed during the development of software systems and contain information at different levels of abstraction. E-mails are used to discuss issues ranging from low-level decisions (*e.g.,* implementation of specific software artifacts, bug fixing, discussing user interface problems) up to high-level considerations (*e.g.,* design rationales, future planning). They can often be written and read by both software system developers and end-users, and always come with additional meta-information (*e.g.,* timestamp, thread, author) that can be taken into account.

Since the FAMIX meta-model was designed to be extensible, adding new information to the source code entities it models is straightforward. However, to add information from e-mails it is first necessary to import messages from the archives in which they reside. Then, the resulting data must be stored in an easily accessible persistent format, in order to be used for subsequent analyses. Finally, the information contained in e-mails must be linked to the entities in the system model, described according to the FAMIX meta-model.

In this article we present *Miler*, a tool infrastructure that tackles these issues and allows one to analyze e-mail archives of software systems. Miler is implemented in VisualWorks[1] Smalltalk, uses the Moose Reengineering Environment for the modeling tasks, and uses GLORP [4] and the Metabase [2] for the object persistency. In previous work we used this infrastructure to create a benchmark for e-mail analysis and to test different lightweight methodologies for linking e-mails and source code [1].

Structure of the paper In Section II we present an overview of the Miler architecture introducing its components and showing how they interact. Then we provide the details of our technique to import archives of mailing lists, and we explain how we store e-mails in a database transparently thanks to a simple meta-model description. In Section III, we discuss how to use Miler to deal with the merging of the model of the system and the model of the e-mail archive. We conclude in Section IV.

II. Miler

Figure 1 depicts an overview of Miler's architecture. After a target system is chosen (point I), the source code is imported into the MOOSE Reengineering Environment through an importer module. The importer parses the source code, generate the corresponding model according to the FAMIX meta-model and exports it in a format that can be loaded by the MOOSE Environment. It is possible both to use a third-part importer or to implement a specific one. For example, when working with Java systems we used iPlasma[2] as importer, while we implemented specific importers to deal with languages (*e.g.,* PHP, Actionscript) not supported by any external tool. After the necessary models are available from Moose, they are included into the core of Miler as "System Models".

After the target system is selected, the e-mails in the archive of mailing lists are also imported into Miler. Different systems use different applications to manage and archive mailing lists, thus not offering a consistent way to access to their data. Moreover, such applications could change during the system lifetime. For this reason, in the worst case scenario, it might be necessary to write at least one importer per system to collect e-mails. We tackled this issue by using MarkMail[3], a free service for searching mailing list archives, which are constantly updated. More than 7,000 mailing lists, taken especially from open source software projects, are stored and displayed in a consistent manner. It is possible both to search e-mails through queries and to access all the e-mails of a specific mailing list. We implemented an importer (Figure 1, point II) that crawls the MarkMail website and extracts all the e-mails from the selected mailing lists and instantiates them

[1] http://www.cincomsmalltalk.com/

[2] http://loose.upt.ro/iplasma/

[3] http://markmail.org/

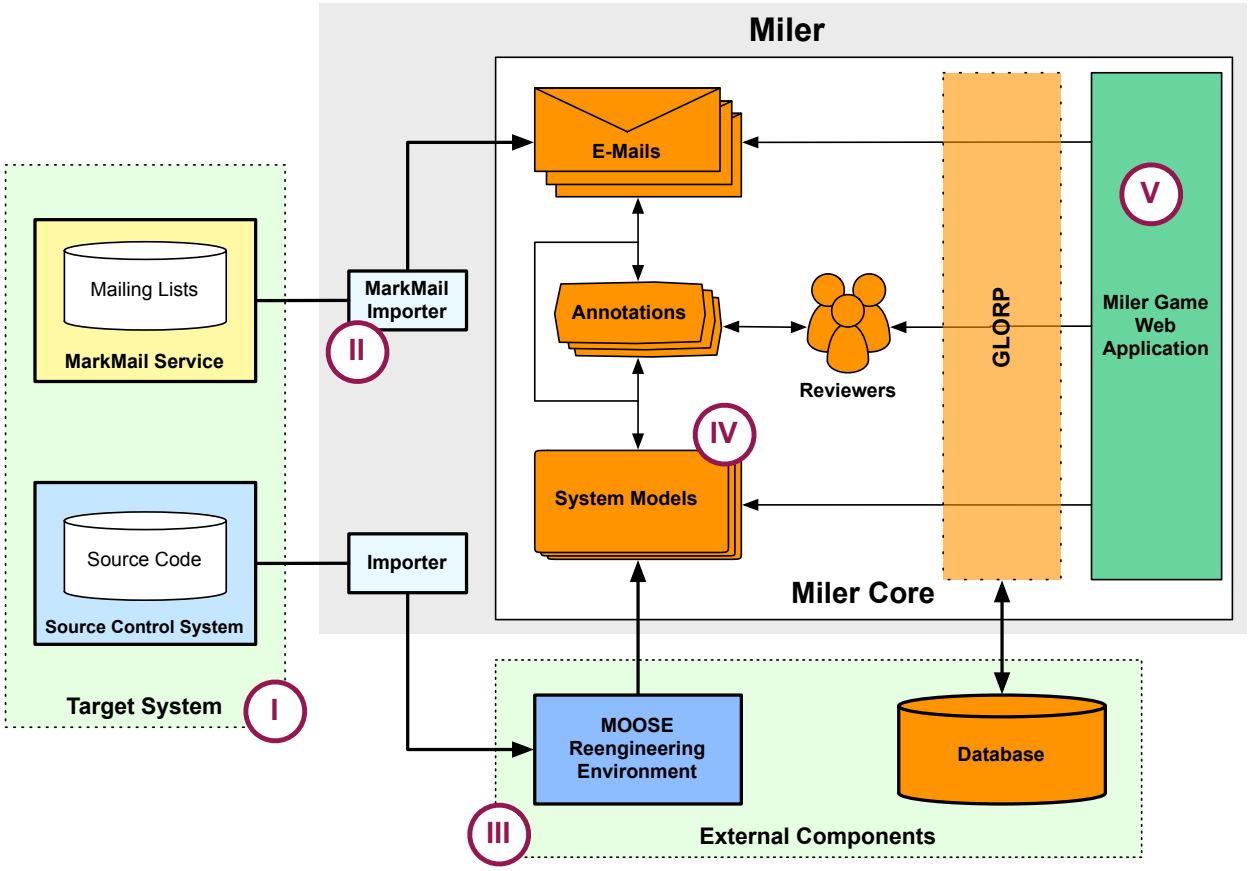

Figure 1. Miler architecture

as objects that are part of the Miler core. As it appears from the architecture diagram, it is easy to add various importers capable of extracting data from other sources than MarkMail service and instantiate them as objects.

To store information gathered from a mailing list, we use an approach based on object persistency rather than using text files (*e.g.,* as done with MOOSE, whose last, and currently used, file format is MSE[4]). Text files do not require a DBMS, however data cannot be accessed remotely, and they generate performance bottlenecks, since the entire text file must always be parsed (*i.e.,* it is not possible to import only parts of the model). When considering mailing lists, the performance aspect is relevant as they often contain thousands of e-mails.

In Figure 1, the orange components that reside inside the core of Miler (Point IV) are modeled according to a meta-model written in Meta[5]. Thanks to these meta-descriptions, the Metabase component is capable of automatically generating the corresponding GLORP class descriptions, which define the mapping between the Smalltalk classes and the database tables [2]. In this way, objects are stored and retrieved from the chosen database transparently through the GLORP layer: It is sufficient to save the objects of the model the first time they are created and to create a connection with the database when loading Miler. In addition, since objects are stored in a common database, it is possible to access them, even remotely, from different languages and applications.

The last component of the Miler architecture is the "Miler Game"(Figure 1, Point V). This is a web application used to manually *annotate* the entities of the systems with the e-mails discussing them. This application is built on the top of the Miler Core using the Seaside web framework [3]. In Section III, we describe in detail the application from a user point of view.

III. Entities and E-Mails

After the core of Miler is filled with the necessary data, gathered from both source code and message archives, the step that follows is extending the FAMIX models with the relevant information that resides inside the model of e-mails, *i.e.,* linking software system entities (*e.g.,* classes) with the

[4]http://scg.unibe.ch/wiki/projects/fame/mse

[5]Meta is the previous version of Fame (http://scg.unibe.ch/wiki/projects/fame/)

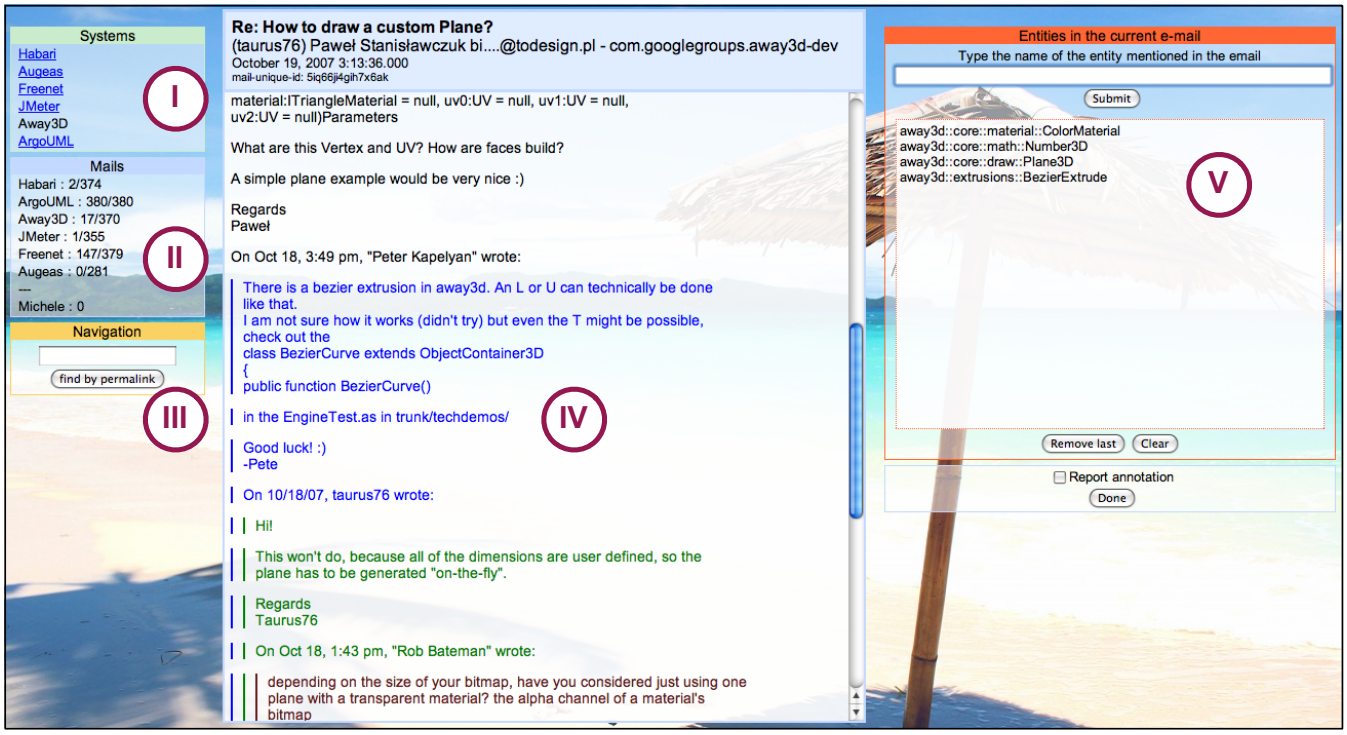

Figure 2. The Miler Game Web Application

e-mails discussing them. Since e-mails are written in free-form text, automatically finding such missing links is not a trivial task [1]. To create a benchmark against which to compare approaches that establish such links, we devised the "Miler Game", a web application that permits to efficiently annotate the system entities with the corresponding e-mails.

Figure 2 illustrates the main page of the web application, which is displayed after the user login. Different interactive panels form this page: the "Systems" panel (Point I) shows the list of the software systems for which both the FAMIX model and the e-mails are available in Miler. The user here chooses the system to be considered. The "Mails" panel (Point II) informs the user about how many e-mails have been read for each system. Since it is possible to setup a predetermined number of mails to read per system (*e.g.,* to create benchmarks [1]), this number is also displayed. The "Navigation" panel (Point III) allows the user to retrieve an e-mail knowing its unique permalink (as we were using the MarkMail importer, we decided to use the one present in the MarkMail service). The main panel of the application is the e-mail panel (Point IV), in which the headers and content of an e-mail are displayed. Headers are displayed on top of the message, including the subject, the author, the date and the list to which the e-mail was sent, and the unique permalink of the message inside Miler. The message content is colored like in common e-mail readers: "Threaded" messages that are part of a larger discussion often quote sentences from previous e-mails, thus, in order to increase readability, the Miler Game colors quoted text differently. Finally, there is the "Annotation" panel (Point V) with two components: the list of the entities that are already annotated (*i.e.,* are discussed in the e-mail) and an autocompletion field (Figure 3).

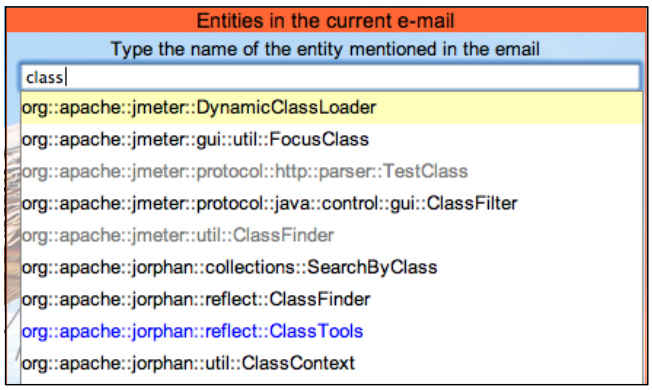

Figure 3. Autocompletion in the Miler Game

This field helps the user in annotating the e-mail at different levels: first, it allows the user to see all the entities whose name include the letters she inserted; second, it avoids typos and forces the user to enter only entities that are really present in the FAMIX models; third, the entity names are colored according to the time proximity of the e-mail

date and the entity release. In Miler, it is possible to have more than one release of the same system, and when the entity names are displayed their date is taken into account. If the entity is present in the last release before the e-mail date, then its name will be colored in black in the autocompletion menu, if the entity is older, then it will be colored in light gray. On the contrary, if the entity appears in the version released after the e-mail date, its color will be blue, otherwise light blue, if present in a version which was released later. This helps the user discerning the appropriate entity. For example, if we consider the class named *ClassFinder*, that is also present in Figure 3, the autocompletion menu shows two different entities with this name: "org::apache::jmeter::util::ClassFinder", in light gray, and "org::apache::jorphan::reflect::ClassFinder", in black. If the class is mentioned only by its name in the e-mail, without the package, and there is no other information, the user can decide to take the latter, as it is more probable that the e-mail is referring to it.

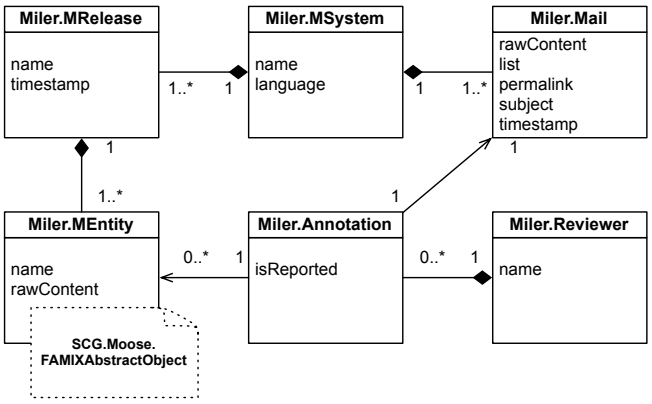

Figure 4. UML Schema of the Miler core

Figure 4 shows the core meta-model behind Miler. *MSystem* is the class representing a system that is imported in Miler. Since GLORP adds an hidden "id" instance variable to every object it stores, it is possible for the name to be not unique. Each *MSystem* owns a collection of *MReleases*, which represent the various version of the source code. Each *MRelease* is characterized by a "timestamp" and has a collection of unique *MEntities*. In our case, we decided to create a new class to represent entities of the system, instead of extending Moose definitions, however it is possible to substitute this abstraction with a *FAMIXAbstractObject* class. To do so, the developer must describe that class meta-model using Meta to specify the information, *i.e.,* instance variables, to be stored and retrieved from the database.

In Figure 4, the class *Reviewer* represents the abstraction of the users of Miler. When a reviewer reads a new e-mail, this generates a new *Annotation* that expresses the reviewer's opinion on the connection between a *Mail* and zero or more *MEntities*. This class shows the "missing link" between source code entities and e-mails. Once the links are validated (*e.g.,* by a review or other expert users), the *Annotation* can be put aside and the *MEntity* can be directly extended with the new information. If *FAMIXAbstractObject* takes the place of *MEntity*, it is possible to extend it either by using a new instance variable or by using the *property* attribute already existing in the class.

Annotations can be generated not only by the manual work of users, but also implementing an automated method. For example, it was shown that searching for entity names into the content of mails using regular expressions is often sufficient to establish a correct link between an e-mail and source code artifacts [1].

IV. Summary

In this paper we have presented Miler, a novel tool infrastructure to establish links between e-mails and source code artifacts. We described the architecture of Miler, discussing the different modules composing it, and presented our implementation and how it can be extended. We then presented Miler Game, a web application we devised for manually linking the information in the e-mails with the entities of the system.

As future work, we plan to extend the "Miler Game" web application to allow selected users to easily perform administrative tasks, such as adding new systems, releases or mailing lists, by simply providing a link to the version control repository or to the mailing list archive.

Acknowledgments We gratefully acknowledge the financial support of the Swiss National Science foundation for the project "DiCoSA" (SNF Project No. 118063).

References

[1] 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 Working Conference on Reverse Engineering)*, pages xxx–xxx. IEEE CS Press, 2009.

[2] M. D'Ambros, M. Lanza, and M. Pinzger. The metabase: Generating object persistency using meta descriptions. In *Proceedings of FAMOOSR 2007 (1st Workshop on FAMIX and Moose in Reengineering)*, 2007.

[3] S. Ducasse, A. Lienhard, and L. Renggli. Seaside: A flexible environment for building dynamic web applications. *IEEE Software*, 24(5):56–63, 2007.

[4] A. Knight. Glorp: generic lightweight object-relational persistence. In *OOPSLA '00: Addendum to the 2000 proceedings of the conference on Object-oriented programming, systems, languages, and applications (Addendum)*, pages 173–174. ACM Press, 2000.