The Harmony Platform

Jean-Rémy Falleri, Cédric Teyton, Matthieu Foucault, Marc Palyart, Floréal Morandat, Xavier Blanc

The Harmony Platform

Jean-Rémy Falleri, Cédric Teyton, Matthieu Foucault, Marc Palyart, Floréal Morandat, and Xavier Blanc

Univ. Bordeaux, LaBRI, UMR 5800, F-33400 Talence, France
{falleri,cteyton,mfoucaul,mpalyart,fmoranda,xblanc}@labri.fr

1 Context and objectives

According to Wikipedia,

> The *Mining Software Repositories* (MSR) field analyzes the rich data available in software repositories, such as version control repositories, mailing list archives, bug tracking systems, issue tracking systems, etc. to uncover interesting and actionable information about software systems, projects and software engineering.

The MSR field has received a great deal of attention and has now its own research conference : `http://www.msrconf.org/`. However performing MSR studies is still a technical challenge. Indeed, data sources (such as version control system or bug tracking systems) are highly heterogeneous. Moreover performing a study on a lot of data sources is very expensive in terms of execution time. Surprisingly, there are not so many tools able to help researchers in their MSR quests [1, 3, 4, 7]. This is why we created the Harmony platform, as a mean to assist researchers in performing MSR studies.

2 Overview of the Harmony platform

The HARMONY platform (`http://harmony.googlecode.com`) has been created to be the Swiss army knife for conducting MSR studies. Whatever your study is, we hope that HARMONY will allow you to set it up quicker than you expected. For this purpose, we designed HARMONY as an highly extensible platform.

Previously, we explained that most of the MSR studies have two main challenges:

- They have to work with a broad set of data sources,
- They perform heavy computation

To cope with these issues, HARMONY includes the following features:

- A simple data model that abstracts the different types of data sources
- A set of sources extractors that can build the abstract model of a broad range of data sources (Git, Mercurial, SVN, CVS, TFS ...)
- A collection of analyses that can be launch on the extracted data models (Object-oriented Metrics,basic statistics, ...).

Of course, each of these three features is extensible, meaning that you can:

- Customize the data model provided by Harmony
- Add new data source extractors
- Develop your own analyses on top of the Harmony model

The cherry on top of the cake is that HARMONY will take care of most of the annoying things, such as dealing with data persistence or exploiting multicore architectures.

3 A unified model

HARMONY provides an unified model that enables you to describe your analysis independently of any VCS. This model is "version" oriented as software evolution is a key dimension in the MSR field. The Figure 1 presents this model.

The `Source` class represents a repository. An `Event` corresponds to a specific revision of the repository. It can have multiple parent events, the HARMONY model is therefore compatible with centralized or distributed versioning systems. Events are made by multiple authors : the `Author` class. Events contain a set of actions (`Action` class and the `ActionKind` enumeration) that can be considered as modifications. Each of these actions are affecting one item (`Item` class), or more precisely a file. We will not go into further details here but be aware that it is possible to extend this general model to fit the need of a specific study. The persistence of all the custom classes will also be handled by the platform, using standard JPA annotations.

Even tough this model is mainly used to abstract source repositories, it was also designed to be compatible with bug-tracking system. That is why the name of some concepts are sometimes vague. For example with a bug-tracking system, an item would be a bug.

4 An extensible platform

The software architecture of HARMONY is based on the OSGi specifications [8] that defines a dynamic component system for the Java language. The Figure 2 details this software architecture.

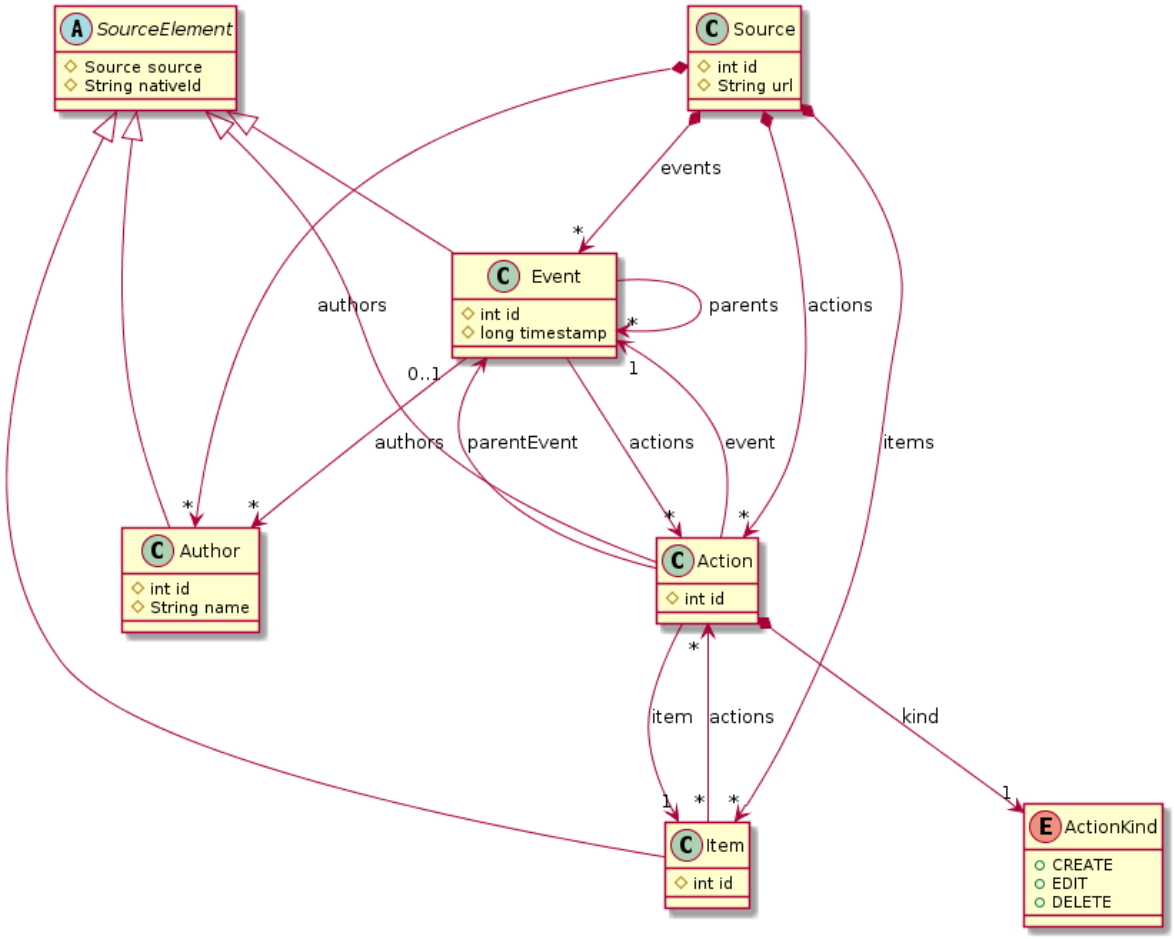

Figure 1: Data model of HARMONY

At the center of the platform is the *core* component that contains the definition of the abstract model, provides the standard features and defines the interfaces of the different services. Among the features provided by the *core* components we find a scheduler which is in charge of executing the analyses in a correct order as well as managing parallelism. The *core* component also handles data serialization to easily save your data model or exchange data between analyses. Finally the *core* component embeds a collection of useful services for dealing with configuration files, output or logging.

The *core* component defines the interfaces of three services:

- IAnalysis: an analysis that takes a source as input. This is the standard way for implementing an analysis. Classes that implement IAnalysis can be chained by specifying the dependencies between them in a configuration file. The scheduler will take care of executing them in a correct order. Data exchanges based on the blackboard pattern [6] can be performed by different analyses.

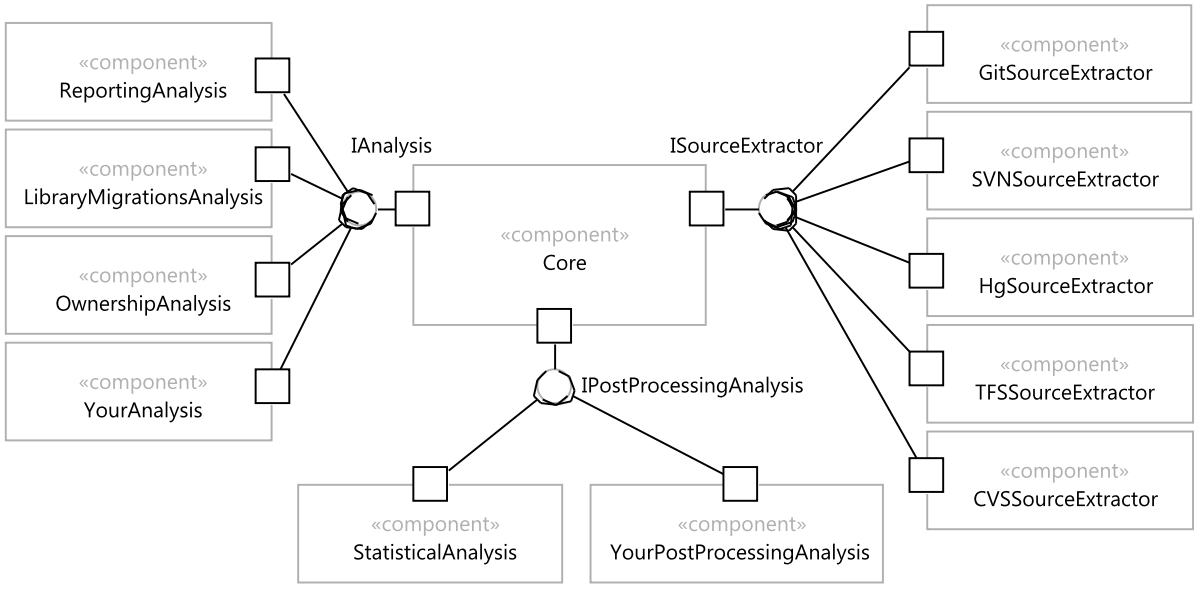

Figure 2: Architecture of Harmony

- IPostProcessingAnalysis: an analysis that take the whole collection of sources as input and that will be executed at the end. There can only be one IPostProcessing-Analysis per study.
- ISourceExtractor: a source extractor is in charge of building the Harmony model by exploring a repository using a particular versioning system.

Thanks to this architecture you can develop an analysis that will be executed on a source repository no matter what versioning system it uses. In addition to the abstract model, the Harmony platform can give access to the repository files in order to perform fine-grained analyses. Developers can then easily benefit from tooling embedded in the Eclipse platform for parsing source code and configuration files such as the JDT[1] or CDT[2].

5 A straightforward tool

Even though Harmony can be used with any OSGi implementation we recommend the use of the Equinox implementation [5] developed by the Eclipse community. That is why we also recommend to use Eclipse as IDE in order to ease the development of your analyses. In this context, we provide an automatic installation procedures as well as a wizard for creating new analyses.

[1]Java Development Tools - `http://www.eclipse.org/jdt/`
[2]C/C++ Development Tooling - `http://www.eclipse.org/cdt/`

```
@Override
public void runOn(Source src) {
  HashMap<Item , HashMap<Author, Integer>> ownership = new HashMap<Item ,
      HashMap<Author, Integer>>();

  for (Item it : src.getItems()) {
    HashMap<Author, Integer> authors = new HashMap<Author, Integer>();
    ownership.put(it, authors);
    for (Action a : it.getActions()){
      for (Author at : a.getEvent().getAuthors()) {
        Integer own = new Integer(1);
        if (authors.containsKey(at)){
          own = authors.get(at)+1;
        }
        authors.put(at, own);
      }
    }
  }
}
```

Listing 1: Example of analysis: computation of ownership

In order to show how easy it is to develop an analysis with Harmony we illustrates it with an example. In the article [2] Bird et al. define that an author is a major contributor of an item if he performed at least 5% of the actions on the files. Otherwise he is a minor contributor. We will now see how to develop an analysis with Harmony that computes the degree of ownership. After installing Harmony and using the wizard for creating a new analysis (see User Manual for details) you will just have to implements the *runOn* method of the analysis class file that was generated for you by the wizard. The listing 1 contains the code needed to compute the degree of ownership for each developer on each file.

6 Perspectives

This papers shows that the current version of the HARMONY platform already enables researchers to focus on designing and running analyses to answer research questions rather than struggling with technical details to implement them. Thanks to the modular software architecture of the HARMONY platform, the situation will carry on to improve with its future versions. Components using various sampling methodologies will be developed to ease the building of representative sets of sources. It will also be possible to embed script based on the R language [9] into analyses in order to chain them directly with standard HARMONY analyses.

References

[1] J. Bevan, E. J. Whitehead Jr, S. Kim, and M. Godfrey. Facilitating software evolution research with kenyon. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 177–186. ACM, 2005.

[2] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu. Don't touch my code!: examining the effects of ownership on software quality. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, ESEC/FSE '11. ACM, 2011.

[3] S. Ducasse, T. Gîrba, and J.-M. Favre. Modeling software evolution by treating history as a first class entity. *Electronic Notes in Theoretical Computer Science*, 127(3):75–86, 2005.

[4] H. C. Gall, B. Fluri, and M. Pinzger. Change analysis with evolizer and changedistiller. *IEEE Software*, 26(1):26–33, 2009.

[5] O. Gruber, B. Hargrave, J. McAffer, P. Rapicault, and T. Watson. The eclipse 3.0 platform: adopting osgi technology. *IBM Systems Journal*, 44(2):289–299, 2005.

[6] B. Hayes-Roth. A blackboard architecture for control. *Artificial intelligence*, 26(3):251–321, 1985.

[7] W. S. Jacek Czerwonka, Nachi Nagappan and B. Murphy. Codemine: Building a software analytics platform for collecting and analyzing engineering process data at microsoft, MSR-TR-2013-7. Technical report, Microsoft Research, 2013. `http://research.microsoft.com/pubs/180138/CodeMine-TR.docx`.

[8] OSGi Alliance. OSGi Service Platform Release 4.3. Technical report, 2012.

[9] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2006. ISBN 3-900051-07-0.