Kenyon-Web: Reconfigurable Web-based Feature Extractor

Sunghun Kim
University of Hong Kong Science and Technology
hunkim@cse.ust.hk

Shivkumar Shivaji, E. James Whitehead Jr.
University of California, Santa Cruz, USA
{shiv,ejw}@soe.ucsc.edu

Abstract

Research on Mining Software Repositories (MSR) has yielded fruitful results in many Software Engineering areas including software change comprehension, bug prediction, and developer network recovery. When performing MSR research, the first task is to extract features corresponding to source code details from repositories. Since reusable feature extraction tools are not available, each MSR research group builds their own extraction tool, a duplication of effort.

We introduce a reusable feature extractor, Kenyon-web, for MSR research. Kenyon-web is fully reconfigurable, pluggable, and serves most MSR related tasks. In this report, we show the architecture of Kenyon-web and demonstrate its utility by showcasing a sample MSR task.

1. Introduction

Recently, Mining Software Repositories (MSR) research has been an active research area within Software Engineering, yielding rewarding results in areas such as software change comprehension [3], bug prediction [7], and developer network recovery [4].

The first step in conducting MSR research is the extraction of artifacts, changes, history, and other information from source code repositories, as part of the feature extraction process. All MSR research groups need a tool to perform such a process. Kenyon [2] is an example of an extraction tool by the Univ. of California, Santa Cruz, and APFEL [6] is another, developed by Saarland University.

Some existing feature extraction tools provide a certain degree of extension, but these are typically very limited. For example, Churrasco et al extract information from software repositories and provide extension points to reuse [5]. However, data is stored in a predefined metadata model and extensions are restricted to data available within. Kenyon-web also supplies predefined data. However, it also exposes raw data such as source code and development history, which provides far more flexibility for extensions. In addition, most previous tools are command line based and require substantial pre-configuration to enable proper use. As a result most researchers in MSR decide to create their own feature extractor rather than reuse existing ones. Thus, wheels are being reinvented again and again.

To provide a reusable feature extractor, we designed and implemented a web-based reconfigurable and pluggable feature extractor, Kenyon-web. Kenyon-web extracts each source code change from repositories and stores information into a database management system (DBMS). After extracting a change, users can add their own plug-ins to perform desirable tasks. For example, users can extract new information from a change and store it in the DBMS for future use.

After extracting all changes from repository, users can add multiple tasks as plug-ins to polish or process useful information from the extracted data. All tasks are configurable through a web interface based on the Hudson build framework [1].

In this demo description, we show the overall architecture of Kenyon-web and pluggable interfaces. In addition, we demonstrate the usefulness of Kenyon-web by performing an example MSR task, change classification [8].

2. Kenyon-Web Architecture

Kenyon-web consists of the SCM repository input, DBMS, revision actor, and after actor interfaces. All interfaces are fully pluggable. The Kenyon-web architecture is depicted in Figure 1.

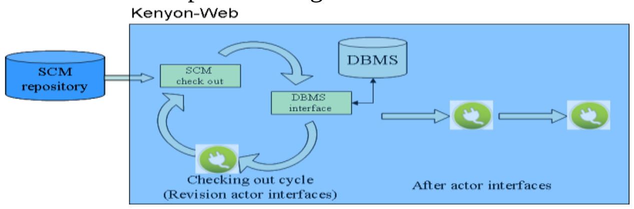

Figure 1. Kenyon-web architecture. The circle of arrows shows the checking out cycle. *Revision actor interface* plugins can extend this cycle. After the cycle, Kenyon-web launches user-defined *after actor interfaces*.

In the basic configuration, Kenyon-web provides Subversion SCM repository access and DBMS interfaces. Kenyon-web checks out revisions from a given SVN repository. For each check out, the DBMS interface reads checked out artifacts and metadata, and writes the data into a DBMS specified by Kenyon-web. Details about DBMS structures and their Java object mappings are available on the Kenyon-web project web page.

As shown in the left cycle in Figure 1, Kenyon-web then launches user pluggable interfaces called *revision actors*. Users can design their own tasks for the checked out revision.

After checking out all revisions and performing DBMS and user defined *revision actor interfaces*, Kenyon-web executes user defined *after actor interfaces*.

3. Pluggable Interfaces

In this section, we explain how to implement and use *revision actor* and *after actor interfaces*.

Figure 2 shows the Java interface for the revision actor interface. Users can simply implement the interface and specify the implemented class to Kenyon-web using its web interface. Then Kenyon will execute the class for each revision check out.

```
public interface IRevisionActor {

public void setUp(KenyonExtractorBuilder kenyonExtractorBuilder,
AbstractBuild<?, ?> build, BuildListener listener, ActorContext context);

public boolean needToCheckout(Transaction<?> tr);

public void visitBeforeCheckout(Transaction<?> tr);
public void visitAfterCheckout(Transaction<?> tr, File workspace);
```

Figure 2. Java interface for the *revision actor*.

An example revision actor would be counting artifacts in each revision. Suppose a user wants to gather statistics on the number of artifact number changes over revisions. He can simply implement the artifact counter as a *revision actor* and plug that into Kenyon-web. Kenyon-web will then execute the counter for every revision.

Kenyon-web supports the *after actor interface*, which is launched after checking out and processing all revisions. This is good for tasks that need complete revision information. For example, suppose we want to draw an artifact count graph after collecting artifact counts from all revisions. This task can be implemented as an *after actor interface*.

Kenyon-web provides the Java *after actor interface* shown in Figure 3. Users need to implement this Java interface and specify the implemented class to Kenyon-web.

```
public interface IAfterActor {

public boolean perform(KenyonExtractorBuilder kenyonExtractorBuilder,
AbstractBuild<?, ?> build, BuildListener listener, ActorContext context,
String args);

}
```

Figure 3. Java interface for the *after actor*.

4. Example MSR Task

To demonstrate the utility of Kenyon-web, this section shows a MSR task involving change classification.

Change classification is an algorithm that identifies a given change as clean or buggy using historical data. To perform change classification, first we need to check out changes from revisions and mark fix revisions. Based on the fix revision marks, we identify buggy and clean changes using annotation graphs. Finally, we extract features from clean and buggy changes. These factors are used to train a machine learner, which classifies changes as clean or buggy. We demonstrate how to perform each task using Kenyon-web.

First, we specify Subversion access information in Kenyon-web. Then we implement fix identification tasks as a *revision interface*. To simplify demonstration, we use a regular expression to match fix revision logs. Suppose there is a 'fix' keyword in the revision log, we mark the revision as a bug fix revision in the DBMS.

We implement two *after actor interfaces*. The first one finds buggy and clean changes based on previous fix revision marks. The second *after actor interface* performs data generation from buggy and clean changes for a machine learner.

We implemented the actors and specify the full implemented class names for Kenyon-web. Kenyon takes care of the rest. Kenyon-web checks out each revision and performs the fix identification task. After checking out all revisions, Kenyon-web first performs the task to identify buggy and clean changes. Finally, it executes the task to generate data for a machine learner. The relevant Kenyon-web is depicted in Figure 4.

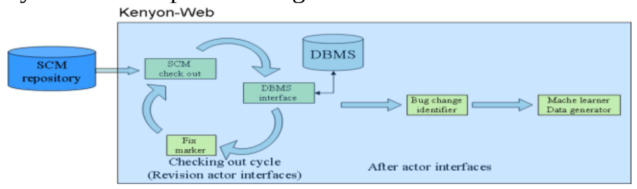

Figure 4. Kenyon-web architecture with user defined interfaces.

5. Conclusions

In this report, we show the overall architecture of Kenyon-web, its pluggable interfaces, and an example of using Kenyon-web for a MSR task. We believe Kenyon-web is reusable for most MSR tasks. In addition to that, the *revision actor* or *after actor interfaces* are also reusable for other researchers, and could be used, for example, to create standard benchmark sets for MSR research. More information about Kenyon-web is available from its project web page, http://slugforge.cse.ucsc.edu/gf/project/kenyon/.

6. References

[1] "hudson: an extensible continuous integration engine," 2009, https://hudson.dev.java.net/.

[2] J. Bevan, E. J. Whitehead, Jr., S. Kim, and M. Godfrey, "Facilitating Software Evolution with Kenyon," Proc. of the 2005 European Software Engineering Conference and 2005 Foundations of Software Engineering (ESEC/FSE 2005), Lisbon, Portugal, pp. 177-186, 2005.

[3] D. Beyer and A. Noack, "Clustering Software Artifacts Based on Frequent Common Changes," Proc. of the 13th IEEE International Workshop on Program Comprehension (IWPC 2005), St. Louis, Missouri, USA, pp. 259-268, 2005.

[4] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu, "Latent social structure in open source projects," Proc. of the 2006 ACM SIGSOFT Foundations of Software Engineering (FSE 2006), Atlanta, Georgia, USA, pp. 24-35 2008.

[5] M. D'Ambros and M. Lanza, "A Flexible Framework to Support Collaborative Software Evolution Analysis," Proc. of Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on, pp. 3-12, 2008.

[6] V. Dallmeier, P. Weißgerber, and T. Zimmermann, "APFEL: A Preprocessing Framework For Eclipse," 2005, http://www.st.cs.uni-sb.de/softevo/apfel/.

[7] A. E. Hassan and R. C. Holt, "The Top Ten List: Dynamic Fault Prediction," Proc. of 21st International Conference on Software Maintenance (ICSM 2005), Budapest, Hungary, pp. 263-272, 2005.

[8] S. Kim, E. J. Whitehead, Jr., and Y. Zhang, "Classifying Software Changes: Clean or Buggy?," *IEEE Transaction of Software Engineering* vol. 34, no. 2, pp. 181-196, 2008.