Integrating Service Oriented MSR Framework and Google Chart Tools for Visualizing Software Evolution

Yasutaka Sakamoto, Shinsuke Matsumoto and Masahide Nakamura
Graduate School of System Informatics, Kobe University
1-1 Rokkodai-cho, Nada-ku, Kobe, Hyogo 657-8501, Japan
Email: gen@ws.cs.kobe-u.ac.jp, {shinsuke, masa-n}@cs.kobe-u.ac.jp,

***Abstract*—We have previously proposed a service oriented framework, named SO-MSR, which applied SOA (service oriented architecture) for conducting the MSR (mining software repository). The principal concept of SO-MSR is to hide complex and complicated mining procedures to end-users for practical use of MSR. Following the SO-MSR, we have also developed MetricsWebAPI which is a web service for metrics measurement. The purpose of this paper is to evaluate the benefits and limitations of SO-MSR and MetricsWebAPI through a development of client system of MetricsWebAPI. To achieve the goal, we develop a consumer mashup application, named MetricsViewer, which integrates MetricsWebAPI and Google Chart Tools. This system is a Ajax web application for visualizing software evolution from a revision control system repository. Through the development experiment, we have confirmed that the SO-MSR enables us easy and rapid implementation of client system, easily integrating with other web services and light-weight execution system.**

***Keywords*-service oriented architecture; mining software repository; visualization; web application; SO-MSR; MetricsWebAPI; MetricsViewer;**

I. Introduction

In the typical software development environment, various version control systems such as source control system and bug tracking system are used to development management. Therefore most of development histories can be recorded in digital data. Currently, a number of researchers have proposed some mining techniques from these software repositories[1][2][3]. By applying these mining techniques, we can obtain valuable knowledge for software development based on empirical historical data.

We have proposed a framework, named *SO-MSR*[4], which applied SOA (service oriented architecture) for conducting the MSR (mining software repository). The principal purpose of SO-MSR is to provide instant and easy way to conduct MSR to general developers who have no specific knowledge about MSR. Generally, conducting the MSR requires non-trivial efforts for data extraction[5], preprocessing for the data and applying mining technique. The SO-MSR introduces an entire process of MSR techniqu as an abstracted cloud service. Mining specific know-how such as complex mining procedure, detailed parameters and accessing protocol for repositories are wrapped in the mining services. MSR practitioner can easily get mining results by invoking service APIs without MSR specific know-how.

Following the SO-MSR, we have also developed a web service with a focus on source code measurement which is one of the major topic in MSR. The developed service, named *MetricsWebAPI*, is a prototype system for measuring source code metrics. MetricsWebAPI wraps and abstracts accessing method of SCS (source control system) repositories and measuring techniques of source codes. User can measure a variety of source code metrics without regard to the difference of SCS repositories and the difference of programming languages. MetricsWebAPI can be accessed over a network using XML-based machine-to-machine protocols such as REST and SOAP.

However since the MetricsWebAPI is just a metrics calculation web service, MetricsWebAPI does not assume end-user's usage and does not support visualization and interpretation of measurement results. The next challenge of SO-MSR and MetricsWebAPI is to develop a client system which wraps MetricsWebAPI.

The purpose of this paper is to evaluate the benefits of SO-MSR and MetricsWebAPI through a development of client system of MetricsWebAPI. To achieve the goal, we develop a consumer mashup application *MetricsViewer* which integrates MetricsWebAPI and Google Chart Tools. MetricsViewer visualizes the software evolution from SCS repositories interactively and intuitively. The advantage of MetricsViewer over other existing visualization systems is that there is no special system requirement. MetricsViewer requires just a network connection and an internet browser.

The organization of this paper is as follows. Section II describes preliminaries Section III presents our proposed MetricsViewer.We discuss the advantages and limitations in Section IV and we conclude in Section V.

II. Preliminaries

A. Challenges in Practical Use of MSR

One of the challenges in practical use of MSR is lack of system or framework to sharing mining techniques and knowledge. Therefore, we need some efforts and technical knowledge for applying MSR.

978-0-7695-4866-1/12 $26.00 © 2012 IEEE
DOI 10.1109/IWESEP.2012.16

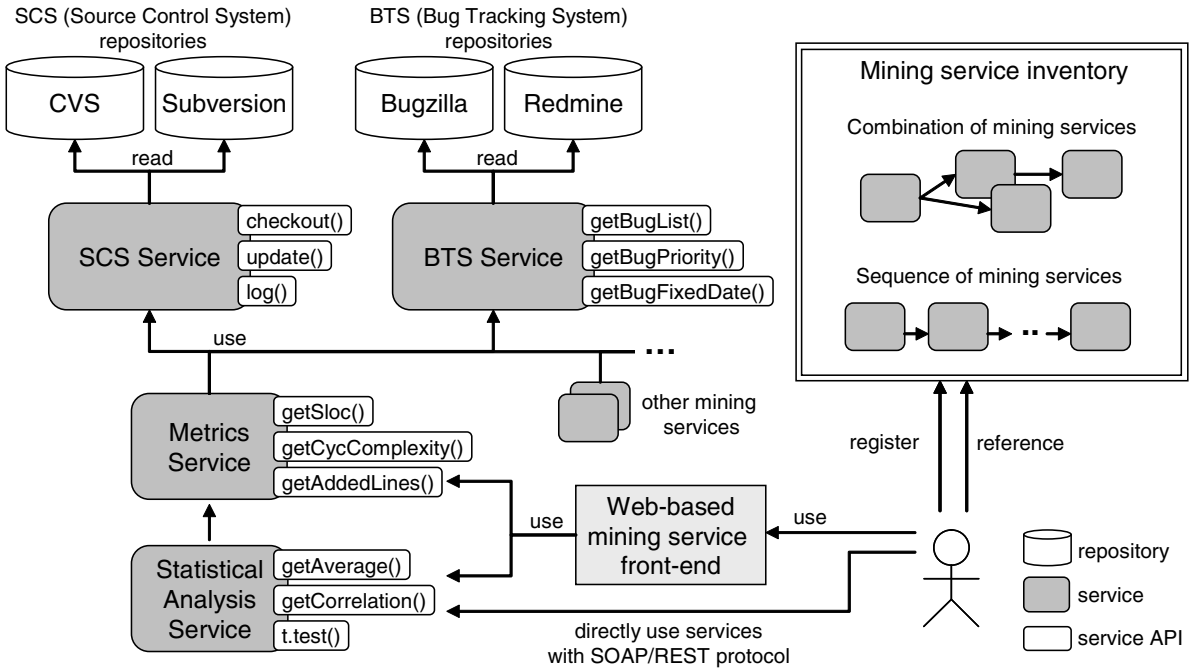

Figure 1. Architecture of SO-MSR

Basically MSR requires following steps; accessing software repositories, quantification (or measurement) of repository data, processing the quantified results. Though wide variety of mining tools has been published on the web, learning effort for each tool and is required for MSR practitioners.

B. SO-MSR

Service Oriented Framework for MSR (SO-MSR) is a framework to support practical use of MSR. The most important concept of SO-MSR is to hide complex and complicated mining procedures for end-users. Following the SO-MSR, mining techniques are wrapped as abstracted services. Each service provides meaningful APIs and abstracts repository access protocol, detailed mining techniques and detailed mining parameters. End-users can easily get mining results without specific MSR knowledge.

Figure 1 shows the architecture of SO-MSR. A cylinder means a software repository, gray-colored box means a mining service and white-colored box means a service API.

Accessing methods of both CVS and Subversion repositories are wrapped in source control system (SCS) service. The SCS service has common SCS APIs such as checkout(), update() and log(). Likewise, bug metrics can be obtained from BTS service APIs without regard to whether the bug is stored in Bugzilla or Redmine. Mining services use these basic repository services. This figure illustrates two example mining services; Metrics Service and Statistical Analysis Service. A user can use the services through a Web application front-end and can directly call the services using SOAP/REST protocol. Service inventory, located in the right-upper, supports sharing a sequence and combination of service invocations. Referring the registered service invocations enables independent validation, mining replication and customizing the mining procedure.

C. MetricsWebAPI

MetricsWebAPI is a Web service for source code metrics measurement which corresponds to "Metrics Service" in Figure 1. Currently, MetricsWebAPI has 32 service APIs, and can calculate 14 types of stati code metrics (i.e., sloc, cyclomatic complexity and C&K metrics) and 12 types of change metrics (i.e., number of code churns, number of developers and number of revisions) directly from two types of SCS repositories (CVS and Subversion).

An example of usage flow of calculation of cyclomatic complexity using MetricsWebAPI is shown as follows. This example uses two APIs with using cURL[1] command with REST protocol.

```
$curl http://metrics.web.api/registerReposi
tory?repository=http://path.to.repository/
<id>1</id>

$curl http://metrics.web.api/getCyclomaticC
omplexity?id=1&code=/src/main.java
<wmc>14</wmc>
```

[1]cURL is a command-line tool to transfer data to or from a server with URL syntax using various protocols.

The first command invokes "regsiterRepository" API with specifying a repository path (i.e., http://path.to.repository/). Its return value means that the registered repository ID is 1. The second command invokes "getCyclomaticComplexity" API with specifying the repository ID and path to target source code (i.e., /src/main.java). Its return value shows that the cyclomatic complexity of target source code is 14.

D. Metrics Visualization for Project Management

Visualization of software repository is a general and popular approach to understand evolutions and histories of a software development to managing the development project. The visualization is intuitively appealing and easily provides "birds-eye view" from a huge software development data. Wide variety of visualization tools and systems have been proposed by a number of researchers [6][7][8][9].

In this paper, we develop "MetricsViewer" which is a web application for visualizing source code evolution. Note that as described in Section I, the goal of this paper is not to propose a novel and useful visualization system but to evaluate the benefits of applying SOA to MSR through the development of the visualization system.

III. MetricsViewer

A. Overview

MetricsViewer is a web application for visualizing software evolution. The MetricsViewer integrates MetricsWebAPI for metrics calculation and Google Chart Tools[2] for metrics visualization. The advantage of MetricsViewer over other existing visualization systems is that there is no special system requirement. MetricsViewer requires just a network connection and an internet browser.

These development style which aggregates and integrates some existing web service APIs is called "consumer mashup"[10]. The benefits of consumer mashup are follows; it does not require extensive programming skills, and saving time for implementing application features.

B. Architecture

The architecture and process flow of MetricsViewer is shown in Figure 2. MetricsViewer is composed of only html and JavaScript.

The visualization process flow is shown as follows.

Step1: A user registers his/her repository through MetricsViewer's html page.
Step2: JavaScript component invokes repository registering API.
Step3: MetricsWebAPI synchronizes the specified SCS repository.
Step4: JavaScript component invokes metrics measurement APIs.

[2]https://developers.google.com/chart/

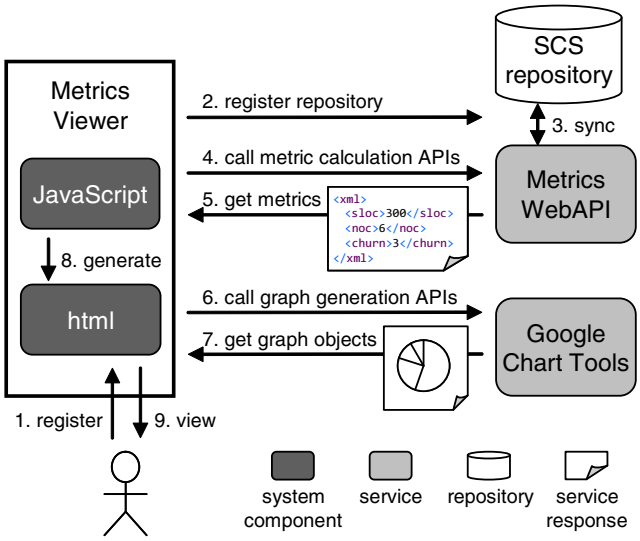

Figure 2. Architecture and process flow of MetricsViewer

Step5: MetricsWebAPI returns calculated metrics.
Step6: JavaScript component invokes graph generation APIs provided by Google Chart Tools.
Step7: Chart Tools returns graph objects.
Step8: JavaScript component generates visualization html page.

C. Features

MetricsViewer has four principal views. A screenshot of MetricsViewer is shown in Figure 3.

Package Explorer

A user selects his/her target source code or target directory stored in a specified SCS repository. This explorer uses `lsr()` API which lists files and directories in the specified repository. In this case, a user specifies "MashMap" project in his/her repository and specifies "Main.java" as a target source code. The following three views are interactively changed by this selection.

Metrics Summary View

This view summarizes some metrics calculated from specified source code or directory. Used APIs are `getLastUpdated()`, `getSloc()`, `getNumberOfContributors()` and so on. In this screenshot, the summarized metrics are calculated from the newest version (revision 28) of main.java.

History View

This view is for visualizing a history of changes of a source code metric for each revision. The y-axis means revision and the x-axis means metric value. The visualized metric can be selected in Metrics Summary View.

Developer View

This view shows activities of developers in target source code or target directory. This screenshot shows that three

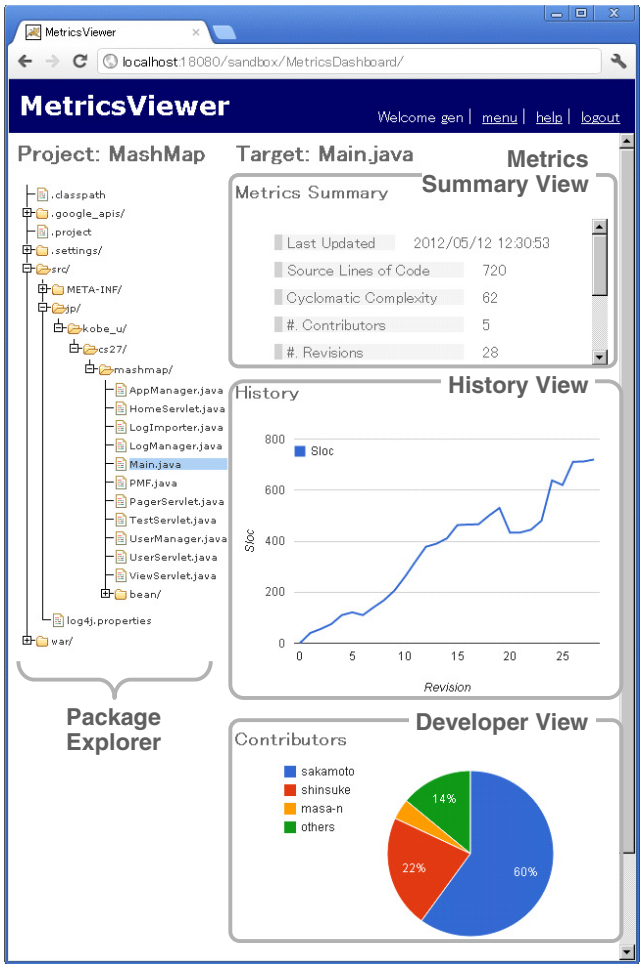

Figure 3. A screenshot of MetricsViewer

main developers and other developers are contributed to the source code.

D. Implementation

We have implemented the MetricsViewer as an Ajax application using HTML5, CSS3 and jQuery 1.7. MetricsViewer can work on Google Chrome and Firefox. A cross domain request of XMLHttpRequest used in the Ajax request is prevented by "same-origin policy" for ensuring browser security. To avoid the same-origin policy, we have introduced CORS (Cross-Origin Resource Sharing) filter to our MetricsWebAPI server. The total lines of code of a JavaScript file was 410, and the development effort was about 1 man-months.

Basically, the MetricsViewer invokes the MetricsWebAPI, parses the XML-formatted results, invokes the Google Chart Tools APIs, and visualizes obtained graph object to a html page.

IV. Discussion

In this section, we discuss advantages and limitations of SO-MSR and MetricsWebAPI through the development experience of MetricsViewer.

Client developer does not required specific knowledge for conducting MSR such as calculation algorithm of each metric. Also the graph generation logics are delegated to the Google Chart Tools. The developer can implement their client application without focus on data processing logics with focus on its interface or application design.

Currently a number of web services are published on the web. MetricsWebAPI also can be used as same as other existing web services. So, MetricsViewer can easily collaborate with other web services and can easily be extended toward more value-added MSR service. In other words, SO-MSR provides benefit in reusability.

Client-side execution environment does not required high computing power and large disk storage. Every mining processes and graph generation processes are executed on a deployed server. MetricsViewer can be used from a low-spec environment such as a mobile device.

However, there are some limitations in SO-MSR and MetricsViewer. First, invoking some web services requires significant network overhead because service request messages and response messages are transferred using XML-based http protocol. This overhead may cause low usability for interactive visualization.

MetricsWebAPI has also a problem in its mining performance. Processing procedures of MetricsWebAPI are (1) accessing a specified repository, (2) applying pre-processing to a specified source code, and (3) measuring a metric. The first step requires network access to the repository, and the second step and the third step need some calculation effort. Additionally, if same request comes again MetricsWebAPI executes same mining procedure because MetricsWebAPI has no caching system,. We need to improve the efficiency of mining process. In our future work, we have a plan to implementation of a caching system using distributed database system.

Mining practitioner can not control detailed mining procedure and parameters because of the abstraction of mining process. For example, cohesion metric LCOM (lack of cohesion of a method) has some varieties[11]. Furthermore Linke et al. have pointed out that measured metric value varies depending on each mining tool[12]. These differences are wrapped and hidden into MetricsWebAPI. Low flexibility and opaqueness of internal process are one of the limitations of SOA.

V. Conclusion

To evaluate the advantages and limitation of SO-MSR and MetricsWebAPI, we have developed a consumer mashup application MetricsViewer which integrates MetricsWebAPI

and Google Chart Tools for visualizing software evolution from a source control system repository.

In our future work, we will continue to develop the MetricsViewer toward personalized metrics visualization system. It will support software developers to instant looking back on their own development activities. Comparative empirical evaluation with other existing MSR techniques must be conducted. Additionally, overcoming the problems of SOMSR and MetricsWebAPI revealed in this study is also the important future work.

ACKNOWLEDGMENT

This research was partially supported by the Japan Ministry of Education, Science, Sports, and Culture [Grant-in-Aid for Scientific Research (C) (No.24500079), Scientific Research (B) (No.23300009)], and Kansai Research Foundation for technology promotion.

REFERENCES

[1] A. E. Hassan, "The road ahead for mining software repositories," in *Frontiers of Software Maintenance*, 2008, pp. 48–57.

[2] A. Hindle, "Green mining: A methodology of relating software change to power consumption," in *Mining Software Repository*, 2012, pp. 78–87.

[3] O. Baysal, R. Holmes, and M. W. Godfrey, "Mining usage data and development artifacts," in *Mining Software Repository*, 2012, pp. 98–107.

[4] S. Matsumoto and M. Nakamura, "Service oriented framework for mining software repository," in *The Joint Conference of the 21st International Workshop on Software Measurement (IWSM) and the 6th International Conference on Software Process and Product Measurement (Mensura)*, 2011, pp. 13–19.

[5] S. Kim, T. Zimmermann, M. Kim, A. Hassan, A. Mockus, T. Girba, M. Pinzger, E. J. Whitehead, Jr., and A. Zeller, "Tare: An exchange language for mining software repositories," in *Mining Software Repository*, 2006, pp. 22–25.

[6] J. Froehlich and P. Dourish, "Unifying artifacts and activities in a visual tool for distributed software development teams," in *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, 2004, pp. 387–396.

[7] C. Collberg, S. Kobourov, J. Nagra, J. Pitts, and K. Wampler, "A system for graph-based visualization of the evolution of software," in *SoftVis '03: Proceedings of the 2003 ACM Symposium on Software Visualization*, 2003, pp. 77–86.

[8] M. Pinzger, H. Gall, M. Fischer, and M. Lanza, "Visualizing multiple evolution metrics," in *SoftVis '05: Proceedings of the 2005 ACM Symposium on Software Visualization*, 2005, pp. 67–75.

[9] L. Voinea, A. Telea, and J. J. van Wijk, "Cvsscan: Visualization of code evolution," in *Proceedings of the 2005 ACM Symposium on Software Visualization*, 2005, pp. 47–56.

[10] S. Mohan, E. Choi, and D. Min, "Conceptual modeling of enterprise application system using social networking and web 2.0 "social CRM system"," in *Proc. Int'l Conf. Convergence and Hybrid Information Technology*, 2008, pp. 237–244.

[11] L. H. Etzkorn, S. E. Gholston, J. L. Fortune, C. E. Stein, D. Utley, P. A. Farrington, and G. W. Cox, "A comparison of cohesion metrics for object-oriented systems," *Information and Software Technology*, vol. 46, no. 10, pp. 677–687, 2004.

[12] R. Lincke, J. Lundberg, and W. Lowe, "Comparison software metrics tools," in *Proc. Int'l Symposium on Software Testing and Analysis*, 2008, pp. 131–141.