Towards Effective Reference Analysis for Software Component Retrieval System

Makoto Ichii† Reishi Yokomori‡ Katsuro Inoue†

†Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan
{m-itii, inoue}@ist.osaka-u.ac.jp

‡ Department of Information and Telecommunication Engineering, Nanzan University
27 Seirei-cho, Seto, Aichi 489-0863, Japan
yokomori@it.nanzan-u.ac.jp

Abstract

Software reuse is broadly recognized as a way to accomplish efficient software development. Especially, lightweight software reuse using software component retrieval systems becomes popular; however, such reuse often causes problems against software traceability since software developers are able to reuse software components without knowing where the components are from. Developers should spend time and effort to investigate the origin of a component to reuse including the ones on which it depends requires additional effort. In this paper, we discuss about problems of current retrieval systems and effective reference analysis between components which aims to help developers reuse components considering the traceability.

1 Software reuse

Software reuse is a popular way to improve software development productivity[7]. But the reused software components may cause legal problems if developers reuse software components without caring the origin or the terms of use of the component. Therefore software reusing activities should be logged as empirical data, which aims to accomplish software traceability and accountability, like other software development activities such as designing, coding, testing and so on. However, collecting the information about reuse is not trivial issue because there are many types of reusable software such as offshore product, open source software in addition to in-house software product.

In this paper, we focus on the lightweight reuse[4], which is contrastive with the controlled reuse[6], using software component retrieval system. In this type of reuse, reuse activity is hard to track because components from various origins are retrieved and reused on-demand. Developers should validate components to reuse whether their terms of use does not violate the policy of the project or company. The origin or author of components may be also required for collecting traceability data. It is desired that software component retrieval systems provide traceability information in addition to components themselves.

2 Software component retrieval system

A software component retrieval system is a system which archives and indexes software components[1]. SPARS-J is a software component retrieval system where developers retrieve Java components by full-text search of source files[5]. SPARS-J provides use-relation (cross-reference) between components and component ranking systems based on the relation. Sourcerer[3] is similar system, while it provides additional features such as fingerprint-based search. Koders[1] supports many types of languages and provides rich origin information, but use-relation is not available.

When a developer reuses a component, usually the components to which the target component has references should be retrieved and reused. However, such referred components are hard to be identified because the database of a software component retrieval system has several components sharing the same name (and the same API). For example, demo.spars.info[2] have three `org.w3m.dom.Document` classes; one of the classes have different source code from the other `Documents` (actually, it seems to be a new version). We found in a preliminary investigation that the 302,545 Java components in the component database of demo.spars.info fall 249,522 groups by their names; 34,642 of the groups contain more than one component; 207 of the groups contain ten and above components. Each of such groups includes versions of a compo-

[1]In this paper, a software component (or a component) means a building unit of a software system such as a module, a function or a class.

nent, different implementations of a same API, or unrelated components sharing a name accidentally.

When a reused component refers another component and a developer needs the referred component, it is enough that a feasible component is automatically identified as the referred component by the retrieval system if he just wants to complete his software system. However, if the developer should be conscious of the origin of reused components, he should investigate the candidates by himself. In other words, the retrieval system should not narrow down the candidates. For example, let component A contains a reference to component B, different components $B1$, $B2$ are named B, a developer may be able to use $B1$ only, while another developer may be able to use $B2$ only.

Therefore, it is needed for software component retrieval system for aiding traceable software development that component use-relation includes alternatives is available in addition to the origin information of components.

3 Reference analysis

For simple, let all classes be referred by the fully qualified name (FQN). A problem where analyzing use-relation between components by identifying the references in a component to other components is described as follows: Let F is a set of references, E is a set of referable programming entities (classes and members of a class in components), $f \in F$ and $e \in E$, the problem is acquiring R which is a set of relation r, relation between f and e. Since we assume the case that there are several classes sharing the same FQN (fully qualified name), in order to make correspondence reference to entity, only one class is selected for each FQN. Therefore each r has an identifying condition D as an attribute. D is a set of $d = (n, e_c, c)$, where n is the FQN of a class, e_c is a class corresponding to n, c represents *confidence* of the correspondence. Confidence represents how a correspondence is credible. The value is an element of $C = \{\text{DEF}, \text{ALT}\}$. The confidence value of d is DEF if there are evidences which denotes that the correspondence should be accurate on the reference; otherwise ALT. Additionally, confidence is also defined to relation: If the condition of the reference consists of DEF only, the confidence of the relation is DEF; otherwise ALT.

Figure 1 depicts an example, where component A has a reference to B and there are two components named B: $B1$ and $B2$. The reference to B is identified and relation is created in two ways. One is the relation to $B1$ with the condition $\{(\text{“B”}, B1, \text{DEF})\}$ because $B1$ is in the same origin with A and it is evidence indicating that A prefers $B1$ rather than $B2$. The other is the relation to $B1$ with the condition $\{(\text{“B”}, B2, \text{ALT})\}$.

When a developer reuse components, the developer considers using the components which have DEF relation first.

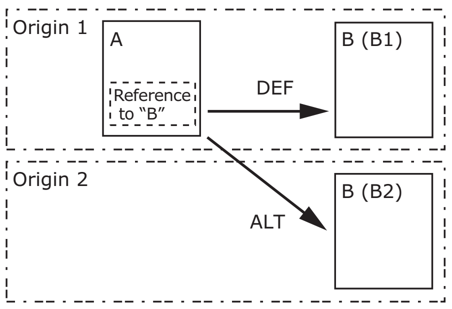

Figure 1. Relation example

If the related component does not satisfy the demand, then selecting ALT relation and retrieves another set of related components.

4 Conclusions

In this paper, lightweight software reuse using software component retrieval system and associated problem against software traceability are described. Then we discussed on use-relation analysis based on identification of references in components in order to encourage lightweight reuse considering software traceability.

We are constructing the brand-new software component retrieval system which takes over the features of SPARS-J and newly implements storing origin information of components and the reference analysis described in this paper.

Acknowledgements This work has been conducted as parts of EASE Project, Comprehensive Development of e-Society Foundation Software Program.

References

[1] Koders. http://www.koders.com/.
[2] SPARS-J. http://demo.spars.info/.
[3] S. Bajracharya, T. Ngo, E. Linstead, Y. Dou, P. Rigor, P. Baldi, and C. Lopes. Sourcerer: A search engine for open source code supporting structure-based search. In *Proc. Int'l Conf. Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'06)*, pages 25–26, Oct. 2006.
[4] R. Holmes and R. J. Walker. Supporting the investigation and planning of pragmatic reuse tasks. In *Proc. 29th Int'l Conf. Software Eng. (ICSE'07)*, pages 447–456, May 2007.
[5] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, and S. Kusumoto. Ranking significance of software components based on use relations. *IEEE Trans. Software Eng.*, 31(3):213–225, Mar. 2005.
[6] I. Jacobson, M. Griss, and P. Jonsson. *Software Reuse*. 1997.
[7] C. W. Krueger. Software reuse. *ACM Computing Surveys*, 41(2):131–183, June 1992.