Reducing the Technical Overhead of Software Reuse

Andrew Z. Tong
Gail E. Kaiser

Columbia University
Department of Computer Science
500 W. 120th Street
New York, NY 10027
Tel: 212-939-7086, Fax: 212-666-0140
tong@cs.columbia.edu

Abstract

This paper discusses three steps (standardization, tailorability and automation) towards reducing the technical overhead of software reuse and describes how these steps are being realized in the on-going Marvel/Oz project of the Programming Systems Laboratory at Columbia University.

Keywords: Componentization; Process Enaction; Process Modeling

Workshop Goals: To exchange information about the current state of automated support for reuse and of reuse within environment technology

Working Groups: Reuse Process Models; Tools and Environments; Domain Analysis

1 Background

In the Marvel project, we developed a multi-user process-centered environment (PCE) that can be reused to define and execute different software development processes for different projects or organizations [1, 2]. The process is modeled as a set of *rules*, each specifying the condition and effects of one software development activity. The process is enforced and automated: by backward chaining to attempt to satisfy the condition of an activity requested by the user and by forward chaining to fulfill the implications of a completed activity [3, 4].

In the successor Oz project, we are developing PCE architectures and components to be reused to produce a family of PCEs as well as members of other environment classes. We defined and executed our own software processes using earlier versions of Marvel to construct later versions. We are now reusing large portions of Marvel's code in constructing Oz, as well as employing a Marvel process that explicitly recognizes and promotes componentization and reuse.

2 Position

Assume a rather ideal situation where there are no legal issues blocking software reuse, no NIH ("Not Invented Here") syndrome, and reuse is encouraged through management support (e.g., when software reuse occurs, both the producer and reuser are rewarded). However, software reuse may still be too costly to be practical, due to its *technical overhead.* This overhead includes those additional activities required to both produce and reuse software, e.g., packaging and classifying reusable software artifacts, searching for and selecting reuse candidates, adapting them to fit the reusers' specific needs, etc. The cost of each of these activities can be prohibitive and hence invalidates the applicability of reusing software.

In the rest of this section, we discuss three incremental steps towards reducing this overhead and briefly describe the approaches we are taking to achieve them in the on-going Oz project.

2.1 Standardization

When trying to reuse a software artifact, the reuser generally has to know:

1. Where to find what he wants.
2. Whether a given software artifact really meets his needs.
3. If so, how to use or reuse it.
4. If it cannot be used without some changes, then how to modify it.

When the software producer and reuser are speaking different "foreign languages" (e.g., using different naming conventions or documentation styles), answering these questions may be rather time-consuming, laborious, and error-prone – if not totally impossible. This task could be considerably easier if both the producer and reuser speak the same "language", or follow the same standards, for example, standards to store and classify software artifacts, to specify functionality, performance, reliability, etc., to define interfaces, and to determine the impacts of changes.

Although these might be difficult to impose across organizations, such languages should be standardized as far as possible *within* an organization – where most reuse is likely to occur – to reduce the overhead of learning a multitude of different "foreign languages" during software reuse.

However, from an individual software developer's point of view, following standards may not be directly beneficial – especially when he expects to move among different companies during his career. Therefore, standards need to become mandatory, like law. To support organization-wide software reuse, standards can be enforced by an organization-wide software development process.

We enforce project-wide software reuse in our Oz project by defining a reuse-oriented software development process that is executed by the existing Marvel PCE. The process is consistently followed by every person involved in the project development; in fact, it is not feasible to access or modify the Oz source code except through this process (unless, of course, one has superuser privileges). This process, called Oz/Marvel, mainly defines activities that belong to the coding phase of the software lifecycle (e.g., how to create, change, and store various software artifacts including source code, run-time libraries, and binaries). We are planning to expand the process support and standardization to areas such as testing, packaging, etc. that would also be useful for our research software.

We have substantial experience using the Marvel PCE to enforce (and automate) our actual software development processes. For example, C/Marvel (C for the C implementation language), was used in Marvel's own development. Doc/Marvel (postscript document production using latex) was used to write the over 400 pages of user and administrator manuals in the last Marvel release, and is also used for thesis proposals, technical reports, etc. P/Marvel (P for Process) is used in developing, testing, installing, and evolving all Marvel processes. Using P/Marvel, the distinction between developing new Marvel processes and maintaining old ones blurs.

2.2 Tailorability

If reusers have to make a tremendous effort to tailor the selected software artifact to fit their specific needs, they may just give up and construct their own from scratch. Tailorability or adaptability of software artifacts requires communication between the software producer and reusers. "Communication" is harder if the pool of potential reusers is not known at the time the software artifacts are produced, so that the producer has to anticipate all possible needs of the reusers (e.g., by performing domain analysis).

In the domain of software process itself, there is as no consensus regarding the best process, and in fact different processes are probably suitable for different projects and organizations.

Consequently, in order to make our Marvel PCE reusable for various process definition and execution purposes, we distinguish the mechanisms for providing services from the policies that govern the behavior of the services. The policies can be easily tailored by what we call the process *administrator*. Specifically, Marvel provides:

- An object-oriented database data modeling (or schema) language, to define the software engineering resources, artifacts, their composition and other relationships, and the information to track the process state [5].
- A pair of process modeling languages, to define process steps and their prohibited, permitted and required sequencing, including a higher-level control-oriented formalism as well as the

underlying rule-based language into which it is translated [6].

- A tool-integration language, to call external off-the-shelf tools to perform the activities of process steps [7].
- A set of lock tables, through which various types of locks and their relationships are defined, to meet the basic concurrency control needs when multiple users participate in the same process.
- A rudimentary coordination modeling language, to express different styles of long duration, interactive, and cooperative transactions representing process segments.

The main theme of our new Oz project is to make the various PCE facilities themselves reusable in designing a *family* of environments, so we separate the PCE architecture from its components. One of the two major components will be a process engine called *Amber*, which serves as a "process virtual machine" that executes a rule-based "process assembly language". The other will be *Pern*, which supports the definition of new concurrency control policies by restructuring (e.g., split, join) in-progress transactions. Our objective is to define interfaces whereby Amber and Pern can be employed as components in several styles of environment architectures, and whereby the Oz architecture can adopt foreign process engines and transaction managers.

2.3 Automation

It is widely believed that significant improvements in software quality and productivity can be achieved only from systematic and comprehensive software reuse – which means that nothing would be started from scratch if there is something can be reused and reuse permeates the entire software lifecycle. Clearly, this introduces much routine work that involves little creativity (e.g., classifying and finding the reusable components). Thus, automation support is essential to making large-scale software reuse practical, that is, software developers should be relieved of this technical overhead to the degree possible in order to be able to focus on the real essence of software design and implementation [8].

However, the applicability of automation depends on the realization of standardization and tailorability. Generally, well-defined standards prove especially amenable to automation [9].

The degree and effectiveness of automation depends on how much knowledge the automation tools have available about the tasks to be automated: the better these tools understand the tasks, the more they can help. This matches well with the objectives of a PCE, which applies the knowledge of the software process to automatically execute portions of it.

We are encoding incrementally into our Oz/Marvel process our increasing knowledge about reusable architectures, components and modules such as encapsulation and functional hierarchy. Specifically, we represent the concepts of system architecture, subsystems, components, etc., using Marvel's object-oriented database schema, and include in the process simple design rules that explicitly carry out automatic reasoning based on these concepts. For example, there are rules enforcing that within a certain system, a component at a given functional level can only invoke visible operations of components at the next lower level, and the detailed implementation of a component is invisible outside that component. And other rules know how to search the component repository to find the proper components and the "glues" to integrate those components into a system.

Separately from Oz, we constructed a special R/Marvel process as an extension of C/Marvel concerned specifically with automating software reuse within a software process. We experimented with modeling and automating some reuse-oriented activities, including classifying and finding the relevant software artifacts. R/Marvel automatically indexes potentially reusable functions based on the naming and documentation standards we followed to develop Marvel. It assists the reuser to find the relevant items by representing queries as objects and providing rules that reason with those objects (i.e., rules concerned with creating and modifying a query).

3 Comparison

Many groups are investigating software process modeling and enaction approaches to formally specify, aid human understanding, and assist in the performance of software processes [10, 11]. Reuse of process knowledge is the main goal of this line of research, whereas reuse of the PCE kernel is a direct consequence.

Several systems apply information retrieval technology to the assembly of large software libraries, based on manually assigning attributes [12] or automatically extracting attributes from natural language documentation. In Guru, a hierarchical clustering algorithm is added to support browsing among query results and closely related reuse candidates [13]. When applied to AIX man pages, the result was encouraging and outperformed the INFOEXPLORER product, in which such a library was built manually. Such tools might be embedded in numerous processes.

Draco is an example of a reuse-oriented system that is based on a hard-wired process [14]. It applies domain engineering, encapsulation and other reuse technologies to guide and automate software development activities within a specific application domain.

References

[1] G. E. Kaiser, P. H. Feiler, and S. S. Popovich, "Intelligent assistance for software development and maintenance," *IEEE Software*, vol. 5, pp. 40–49, May 1988.

[2] I. Z. Ben-Shaul, G. E. Kaiser, and G. T. Heineman, "An architecture for multi-user software development environments," *Computing Systems The Journal of the USENIX Association*, vol. 6, pp. 65–103, Spring 1993.

[3] G. T. Heineman, G. E. Kaiser, N. S. Barghouti, and I. Z. Ben-Shaul, "Rule chaining in MARVEL: Dynamic binding of parameters," *IEEE Expert*, vol. 7, pp. 26–32, December 1992.

[4] N. S. Barghouti, "Supporting cooperation in the MARVEL process-centered SDE," in *5th ACM SIGSOFT Symposium on Software Development Environments* (H. Weber, ed.), (Tyson's Corner VA), pp. 21–31, December 1992. Special issue of *Software Engineering Notes*, 17(5), December 1992.

[5] N. S. Barghouti and G. E. Kaiser, "Modeling concurrency in rule-based development environments," *IEEE Expert*, vol. 5, pp. 15–27, December 1990.

[6] G. E. Kaiser, S. S. Popovich, and I. Z. Ben-Shaul, "A bi-level language for software process modeling," in *15th International Conference on Software Engineering*, (Baltimore MD), pp. 132–143, IEEE Computer Society Press, May 1993.

[7] M. A. Gisi and G. E. Kaiser, "Extending a tool integration language," in *1st International Conference on the Software Process: Manufacturing Complex Systems* (M. Dowson, ed.), (Redondo Beach CA), pp. 218–227, IEEE Computer Society Press, October 1991.

[8] F. P. Brooks, Jr., "No silver bullet: Essence and accidents of software engineering," *Computer*, vol. 20, pp. 10–20, April 1987.

[9] G. Caldiera and V. Basili, "Identifying and qualifying reusable software components," *Computer*, vol. 2, pp. 61–70, February 1991.

[10] I. Thomas, ed., *7th International Software Process Workshop: Communication and Coordination in the Software Process*, (Yountville CA), IEEE Computer Society Press, October 1991.

[11] *2nd International Conference on the Software Process: Continuous Software Process Improvement*, (Berlin, Germany), IEEE Computer Society Press, February 1993.

[12] R. Prieto-Diaz and P. Freeman, "Classifying software for reusability," *IEEE Software*, vol. 4, pp. 6–16, January 1987.

[13] Y. S. Maarek, D. M. Berry, and G. E. Kaiser, "An information retrieval approach for automatically constructing software libraries," *IEEE Transactions on Software Engineering*, vol. 17, pp. 800–813, August 1991.

[14] J. M. Neighbors, "The Draco approach to constructing software from reusable components," *IEEE Transactions on Software Engineering*, vol. 10, pp. 564–574, September 1984.

[15] I. Z. Ben-Shaul, "Oz: A decentralized process centered environment," Tech. Rep. CUCS-011-93, Columbia University, Department of Computer Science, April 1993. PhD Thesis Proposal.

[16] G. T. Heineman, "A transaction manager component for cooperative transaction models," Tech. Rep. CUCS-017-93, Columbia University, Department of Computer Science, April 1993. PhD Thesis Proposal.

4 Biography

Zhongwei Tong is a PhD candidate in the Columbia University Department of Computer Science. His research interests include software development environments, software process, and reuse. He received his BS from Jiao Tong University, China and his MS from Columbia University.

Gail E. Kaiser is an Associate Professor of Computer Science and Director of the Programming Systems Laboratory at Columbia University. She was selected as an NSF Presidential Young Investigator in Software Engineering in 1988. Prof. Kaiser has published over 80 papers in a range of areas, including software development environments, software process, cooperative transactions, testing and debugging, reuse, application of AI to software engineering, object-oriented languages and databases, and parallel and distributed systems. Prof. Kaiser is an associate editor of the journal ACM Transactions on Software Engineering and Methodology, and serves on numerous program committees for conferences as well as reviewing for conferences, journals, and the NSF. She received her PhD and MS from CMU and her ScB from MIT. She is a member of AAAI and ACM and a senior member of IEEE.