JCMP: Linking Architecture with Component Building

Guoqing Xu gqxu_02@cs.ecnu.edu.cn
Software Engineering Lab, Department of Computer Science
East China Normal University, Shanghai 200062, P. R. China

Abstract:

Components defined in the software architecture differ from common components in the fact that they must conform to the architectural conformance. On the other hand, as the independent modules, they should also be developed individually without dependency on each other for the better reuse.

The Problem

However, these two issues have always been handled separately without any relations. Existing approaches to solving one of these two problems can be no longer available when facing another one.

Methods for keeping architectural conformance:

In order to enable architectural reasoning about an implementation, the implementation must conform to its architecture. One of the famous works is the three criteria identified by Luckham and Vera [LV95] for architectural conformance. They are rules of Decomposition, Interface Conformance, and Communication Integrity. According to these three criteria, researchers have developed a wide variety of Architecture Definition Languages (ADLs) to describe, model, check and implement software architectures [MT00]. In these languages software connectors are recognized as an important consideration to adapt one component's interface to the interface of another [SDK05+]. One of the good examples is ArchJava [ACN02]. ArchJava unifies architectural abstraction and detailed implementation in one language, using type to enforce the architectural conformance, especially the communication integrity in the implementation. In the system of ArchJava, based on the communication integrity criterion, component A directly invokes B's provided method m if and only if their interfaces are connected in the architecture.

However, we notice there is the direct client-server relationship between A and B. This direct invocation requires that the "requires" method declared in A's interface must have the same name and signature with the "provides" method in B's interface. That is if B evolves changing the name of method from *m* to *n*, or B is replaced by a new component D with it providing a method *p* which has the same function and arguments type with *m* of B, the "requires" interface of component A must also be modified manually.

If both concrete components A and B have existed and been built by different software houses, can they be reused in the architecture? The answer might be no since the existing ADLs require the signature of required-provided methods pairs must be completely same. They only focus on architectural conformance problems without considering how to separate the whole architecture into independent reusable building blocks and the integration of pre-build generic components into the architecture-based application. This problem may greatly hinder the component reuse.

Methods for composite adaptation:

Generally speaking, there are two major kinds of approaches to supporting system composition from individual modules. One is to use Module Interconnection Language (MIL) which describes the uses relationship between components, such as Jiazzi [MFH01], a component infrastructure for Java and a similar system, knit for component-based programming in C. However, MILs can not be used to describe the architecture and therefore, this kind of implementation does not support architectural reasoning. Another kind of approaches is like Pluggable Composite Adapters (PCAs) [MSL01] and their predecessor, Adaptive Plug and Play Components (APPCs) which offers different means for on-demand remodularization. The on-demand remodularization means the abstractions and vocabulary of an existing code base are translated into the vocabulary understood by a set of components that are connected by a common collaboration interface. However, this approach has not considered the architectural constraints, and communication integrity can not be enforced.

From the above discussion, we notice these two issues are totally handled separately. However, if the generic components composition is not supported, software architecture will be far from practical. On the other hand, if architectural conformance can not be enforced, the architectural definition will be meaningless in the components development. These two problems are far from orthogonal although what they concern is different.

Our Approach

We try to band these two problems together for consideration and put forward triple-C model to solve them. Triple-C model stands for Components-Communicate-through-Connector. Triple-C model is derived from the three criteria in [LV95], but there are two major differences between them. One is that in triple-C model, there should be not only a concrete component in the implementation for each one defined in the architecture, but also a concrete connector for each abstract connection. Another one lies with the change of definition of communication integrity. Since we have discussed the harm resulting from the direct client-server relationship in components communication, in triple-C model, a component can only communicate with another one through the invocation transfer performed in the connector if they are connected in the architecture. Since in triple-C model, the implemented connector is the direct client and server of two connected components, components are able to be built independently without knowing the detailed methods they want, which supports the integration of generic components. On the other hand, because the connectors are also elements in the architecture, this implementation better reflects the architecture definition, therefore helps architectural reasoning and keeps the architectural conformance.

We have designed a novel ADL JCMPL and a toolset JCMP to help apply the triple-C model. JCMPL only defines abstract architecture, without any detailed type operations. JCMP toolset includes five independent tools: JCMP/Compiler, JCMP/Kernel, JCMP/Match, and JCMP/Checker. JCMP/Compiler compiles the architecture abstraction defined by JCMPL and automatically translates it to Java implementation. Especially, it can translate connector specification to connector implementation, which is used to transfer the invocation between components. JCMPL/Kernel contains the prototypes of Primitive Component, Advanced Component, port, interface and connector which are to be extended by implementation; JCMP/Match performs the automated methods matching in the two connected ports, based on the program invariants; and JCMP/Checker finally checks the validity of implementation codes and its conformance with architecture.

What we desire to poster are the overview of JCMPL language, JCMP toolsets and detailed techniques used in my system to both enforce communication integrity and enable composite adaptation.

Reference

[LV95] David C. Luckham and James Vera. An Event Based Architecture Definition Language. *IEEE Trans. Software Engineering 21(9)*, September 1995.

[MFH01] Sean McDirmid, Matthew Flatt and Wilson C. Hsieh. Jiazzi: New-Age Components for Old-Fashioned Java. *Proc. Object Oriented Programming Systems, Languages, and Applications*, Tampa, FL, October 2001.

[MSL01] M. Mezini, L. Seiter, and K. Lieberherr. Component I ntegration with pluggable composite adapters. In M. Aksit, editor, Software Architectures and Component Technology: The State of the Art in Research and Practice. Kluwer, 2001. University of Twente, The Netherlands

[MT00] Nenad Medvidovic and Richard N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Trans. Software Engineering*, 26(1):70-93, January 2000.

[ACN02] Jonathan Aldrich, Craig Chambers and David Notkin. *Proc. International Conference on Software Engineering*, Orlando, FL, 2002

[SDK05+] Mary Shaw, Rob DeLine, Daniel V. Klein, Theodore L. Ross, David M. Young, and Gregory Zelesnik. Abstractions for Software Architecture and Tools to Support Them. *IEEE Trans. Software Engineering,* 21(4), April 1995.

Author’s Short Biography:

Guoqing Xu hodes the B.Eng from East China Normal University. He is a second year Master student under the advisory of Prof. Zongyuan Yang in Department of Computer Science and a research assistant in the Software Engineering Lab (SEL) of East China Normal University. His primary technical interest is software reliability and software reuse which span the spectrum fromprogramming languages, through program analysis, to software engineering. He is a student member of ACM and its special interest group on software engineering (ACM SigSoft). He is also a student member of Shanghai Computer Society (SCS).

Email: gqxu_02@cs.ecnu.edu.cn
Phone (home) : 86-21-5268-9731 (Office): 86-21-6223-3654
(Fax): 86-21-6286-1049
Advisor: Zongyuan Yang, Professor and Chair of Department of Computer Science, East China Normal University
Email: yzyuan@cs.ecnu.edu.cn Phone: 86-21-6223-2642 (Office)