A Basic Model for Components Implementation of Software Architecture and the Language and Tools to Support it

Guoqing Xu
Software Engineering Lab, Department of Computer Science, East China Normal University
Shanghai 200062, P. R. China

Software architecture describes the high-level structure of a software system, and can be used for design, analysis, and software evolution tasks. A number of researchers have developed a wide variety of Architecture Definition Languages (ADLs) to describe, model, implement software architectures and check the architectural consistency in the implementation.

Existing ADLs can be mainly classified into two categories from the ways they use to handle the relationship between the architecture and implementation. One kind of them decouple architecture from implementation, allowing inconsistencies to accumulate as a software system evolves. Because of the potential for inconsistency, engineers evolving a program cannot fully trust the architecture to accurately describe the properties or structure of the implementation. Another kind of ADLs, exemplified by ArchJava, seamlessly integrate the Software Architecture and the detailed implementation, by unifying them in one language. In this kind of ADLs, a strong type system has been used to keep the architectural constraints in the implementation. That is they use language-level types to enforce the architectural structure. However, these ADLs can be provided as the design language for the software architect designing the overall system, since the detailed codes can never available in the design phase.

There is another problem existing in the current research on the Software Architecture. Components defined in software architecture have two features: as basic elements of the architecture, they must conform to the architectural constraints and in the meantime, similar to the common components, they should have the flexibility to be developed independently for the late third party integration. However these two issues are totally handled separately without any relations. 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. That is these two problems are far from irrelevant although what they concern are different.

This paper presents a basic model of the architecture-based components implementation to band these two issues together. It firstly describes a novel design pattern, triple-C pattern which stands for Components-Communicate-through-Connector. This pattern not only emphasizes that implementation must completely conform to the architectural definition, but also makes the attempt to change the fundamental way of components communication with suggesting provided service should be transferred through the connector instead of directly between the client and server components. Second, it describes a novel ADL JCMPL, toolset JCMP and techniques to keep architectural conformance in the implementation as well as support the architectural integration from separate components. Since triple-C pattern based implementation better reflects the architecture definition, it also presents techniques to dynamically discover the architectural abstraction from the implementations. Finally, this model is evaluated in a case study.