Challenges in Implementing Software Architectures

Marija Rakic Nikunj Mehta
{marija, mehta}@sunset.usc.edu
Department of Computer Science
University of Southern California
Los Angeles CA 90007

Background

C2 is a component and message based architectural style. The key ideas for C2 came from the experience with Chiron-1 user interface system and other styles (Unix pipe-and-filter, blackboard, etc). The most important elements of the C2 style are components and connectors [1]. In particular, the style requires that components cannot directly interact with other components, but must do so via connectors. C2 treats connectors as first class entities, but supports a small number of connector types, mainly message passing and implicit invocation.

Figure 1 shows the object-oriented framework we designed for developing architecture implementations in the C2 style. The C2 framework has so far been implemented in variety of programming languages (C++, Java, Ada, Python) and is being evolved for use on embedded devices. This base framework has been extended to provide inter-process communication through the use of various middleware technologies [3].

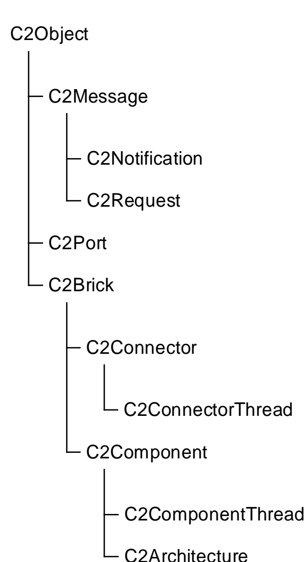

Figure 1 Object Oriented Framework supporting C2 architecture implementation

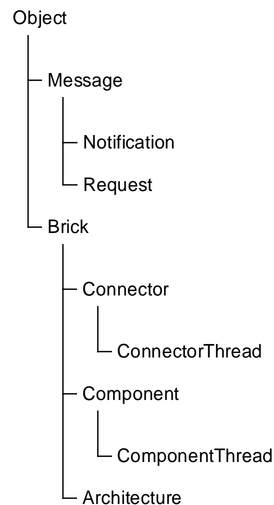

Figure 2 Modified object-oriented C2 framework

Improving the Implementation Infrastructure

The object-oriented framework in Figure 1 has been used to develop various applications intended to demonstrate the feasibility of architecture-based software development according to the rules of C2. We have used these experiences to evaluate the framework and determine its limitations. An extensive analysis of the framework indicated that ports, which handle message

queues for each architectural block, are not required as first-class entities and tend to lose their significance once an architecture is refined into a design and/or implementation. On the other hand, the C2 connectors, which were treated as solely message carriers, are now looked at as being more integral means of interaction between components, including security, fault-tolerance, and distribution.

Figure 2 shows a simplification and improvement to the framework to support more types of interaction and a more meaningful transition from architecture to implementation. The main changes to the original framework are that *Architecture* is derived from a *Brick* instead of a *Component*, thus allowing the connectors to have arbitrarily complex internal architectures, and that the *Ports* are no longer explicit entities in the implementation. Our initial studies suggest that the modified framework preserves the original framework's flexibility, while improving application performance and reducing size. At the same time, further study is needed to help us determine what architectural elements are most useful from both a theoretical and a practical point of view.

The current C2 framework only supports the broadcast of simple messages to all attached components in the architecture. However, many practical systems involve the direct interaction between components and require large amounts of structured data to be communicated. To address this problem we have restructured the framework to support composition of complex connectors. We have developed taxonomy of software connectors to understand the gamut of component interactions [4]. This taxonomy also indicates the areas where C2 is weak in providing component interactions, such as security, transactions, message distribution, stream based communication, etc. Although messages form the bulk of component interactions, inefficiencies such as message cloning for dispatch and message broadcast affect the system performance significantly. In order to provide a richer set of component interactions, we are implementing a message distribution connector similar to data transfer in a network. We have also begun implementing security connectors, multi-way procedure calls, and system gauges.

The Big Picture

Apart from the need to have functional elements in the architecture, it is necessary to ensure that an architectural framework is flexible and efficient. This issue has become more prominent as we have attempted to apply the C2 style to embedded systems and handheld computing. The question we are facing is: What is the right balance between framework efficiency and flexibility? Our argument is that, as researchers, a primary goal is to explore the unchartered and hence the uncertain. Architectures are haute, but to make them more useful, we have to provide innovative services that go beyond the normal primitives. In the past new primitives such as RPC, threads and callbacks were identified through experimentation and these concepts have been extended over time into more powerful and less expensive frameworks (e.g. the evolution of DCE RPC to CORBA and now to Java RMI). The architecture community should therefore push forward on providing infrastructure to conduct research that concocts new primitives and provides means to handle increasing complexity of software. Simultaneously, we should work on optimizing the implementation of architectural constructs by simplifying the implementation frameworks and transforming architecture constructs into efficient code modules. Computer systems research (e.g. distributed systems, networking) can be used as a starting point for identifying new architectural elements.

The purpose of software architecture frameworks is to support the development of software in a disciplined manner so that architecture descriptions can be converted to real systems. Architecture-based development should lead us to a discipline based on handbooks, guides and standards rather than the current choice between intuition and cryptic math that only a few can understand. The average developer is not likely to use tools to prove architectural properties as much as using infrastructures to build systems and handbooks that provide guidance to designing

solutions based on already proven architectures. This paper has discussed such an infrastructure that software developers can employ in implementing systems. The key property of the infrastructure is its preservation of architectural constructs and characteristics in a system's implementation. We are continuously investigating techniques to minimize the costs (size, performance, flexibility) incurred by the use of the explicit architectural constructs. One such technique has already resulted in a simplification and optimization of the infrastructure.

References:

[1] Taylor et al., A Component- and Message-Based Architectural Style for GUI Software. IEEE Transactions on Software Engineering, June 1996.

[2] Taylor et al., Chiron-1: A Software Architecture for User Interface Development, Maintenance, and Run-Time Support, ACM Transactions on Computer-Human Interaction, June 1995.

[3] Dashofy, Medvidovic and Taylor, Using Off-The-Shelf Middleware to Implement Connectors in Distributed Software Architectures, Proceedings of ICSE 99, May 1999.

[4] Mehta, Medvidovic and Phadke, Towards a Taxonomy of Software Connectors, Proceedings of ICSE 2000, June 2000 (to appear).