Architecture or Component Frameworks?

Alexander L. Wolf
Department of Computer Science
University of Colorado
Boulder, CO 80309-0430 USA
alw@cs.colorado.edu

I have been involved in the field of software architecture since the late 1980s when it became recognized as a distinct discipline and emerged from the general area of software design. Since that time I have done what I could to promulgate the field, preaching its "obvious" truths and benefits:

> A good system has a clear, well-defined structure that neatly separates functionality into tractable units. If the structure is captured as a design artifact, then you can do things such as analyze the design for various properties that serve as reliable predictors for corresponding properties of the implementation. If the structure is captured as an implementation artifact, then you can do things such as safely reconfigure the system within a known set of acceptable constraints. If the structure is captured in both design and implementation artifacts, and if the consistency of those artifacts can be guaranteed, then you can do things such as reliably assess and predict the impact of making structural changes to the system. Moreover, you can use the design artifact as an abstraction of the implementation artifact suitable for such things as contracting and training.

To bolster this view, most of the energy in software architecture research during the past ten years has gone toward creating formal notations, design environments, and assessment practices for software architects. This activity has gone on largely in parallel with, and independent from, the effort that has gone into the development of component frameworks such as CORBA and its cousins. Has the energy put into software architecture research paid off? Consider the following questions:

- Can I, today, reduce my risk more by investing in the development of a good architecture or by adopting a good component framework?
- Can I, today, better predict the scalability, reliability, security, deployability, or maintainability of my system by assessing a description of an architecture or by reflecting on the body of experience with an existing component framework (as well as the individual components fitting into that framework)?
- Whose skills are considered more valuable today, the architect or the component programmer?
- Can I, today, make more money selling an architecture or a component that fits into an existing component framework?

We would like to believe that the answer to these questions is architecture, but in reality it is component frameworks.

The fundamental issue is that an architecture is more than the sum of its parts. At least, it *should* be or there is no value to be gained from attending to architectural concerns above and beyond those of component concerns. Unfortunately, there is currently a higher cost to developing, maintaining, and analyzing architectures than there is benefit to having them.

Compare this to what proponents of component frameworks have to offer. Generally speaking, they provide a ready-made solution to the challenges of modern, complex software development: Their frameworks inherently support distributed systems, they are compatible with popular object-oriented design and implementation techniques, and they are supported by commercial products. It would be hard to argue that support for software architecture comes close.

The counter argument is that component frameworks support only a relatively specific architecture, while the field of software architecture is trying to address more general structures. This is quite reasonable. But if that is the case, then software architecture technology should in some way be useful to the developers and users of component frameworks. To my knowledge, there is no evidence that there has been any such contribution.

Which leads to a proposal for a software architecture challenge problem or, at least, problem area. The challenge is to demonstrate the ability for software architecture technology to support or enhance component framework technology. Some possible tasks include the following.

- Modeling and then analyzing (since modeling for its own sake is quite nearly worthless) component frameworks as software architectures.
- Capturing and supporting the use of a component framework as an architectural style.
- Providing a rigorous method to choose among alternative component frameworks when developing a particular system.
- Showing how to integrate some software architecture technology into an existing component framework support system (e.g., static analysis of behavioral compatibility among components or dynamic reconfiguration of a system of components).

Theoretically, these things are doable, but are they valuable? This is the critical, difficult next question to answer.

Sadly, the situation involving software architecture and component frameworks reminds me of the situation that exists between software process research and workflow research. Software process research took the “high road” of generality and broad applicability, while workflow research took the “low road” of attacking a particular domain. This strategy has allowed workflow technology to gain a viable constituency and a large amount of credibility, something that software process research has so far failed to achieve. On the other hand, software process research has led to solutions to problems that are only now being encountered by workflow researchers who are trying to expand their domain. One can make the assertion that this is true of software architecture research as well, but the burden of substantiating this assertion lies with software architecture researchers, not their friends in the component framework world.