

Component Based Software Engineering - Short Paper Session -

Asuman Suenbuel
Kestrel Institute,
3260 Hillview Avenue, Palo Alto, CA 94304
asu@kestrel.edu

We are computer scientists, we love to play around with code, and most of us have developed our own language. A colleague of mine -a computer scientist- told me how he spent the entire night before his wedding debugging a wine tasting program that he wrote for his party.

I have observed in all of the computer science institutes I've been affiliated with, that we struggle for something, whatever this is, it steals a large amount of our time. Often, I met around midnight, colleagues in their offices busy with coding, testing, and whatever other things "normal" people wouldn't do at such a late, or rather early hour. Some of us have even experienced two sunrises on a single working day! But what are we actually doing? We are most of the time formulating rules for automating daily routines for ourselves and/or our users. Therefore, we have to be familiar with every detail and exception occurring in our system routines. We indeed start at a very low level and try to implement a system. Is this procedure not called "fine tuning" in other disciplines?

When we look at how bridges, houses or cars are being constructed, we notice a difference in how these disciplines apply rules to attain the target product.

First, an architect or an engineer describes the frame and provides rules that have to be applied. Analyzing their way of building "systems", we notice, that there are core functionalities and frames, which remain quite stable. Other parts are subject to change. There are issues of fine-tuning and detailed design and the materials being used are usually known: possible side effects, advantages and disadvantages are a priori clarified.

Certain analogies apply as well to building software systems, which we categorize as system invariances:

The first order invariance: has the highest inviolability degree. First order invariant changes are the most "radical" changes of the software system. This may involve a total restructuring of the whole system architecture.

Second order invariance: The second order invariance concerns the connectors and the incarnation of the connectors. Connectors realize the interaction among components. The second order invariant changes are

those, for example, that modify the connectors wrt. protocols and platforms. This degree of invariants is mainly used, if renovations of the regarded system must be done.

Third order invariance: The third invariance degree describes system modifications requiring adjustments in other parts of the system different from those where the actual changes have been made. These kinds of changes are, for example, the modification of data structures and operations being visible at the export and/or import interface of a component.

Fourth order invariance: Modifications belonging to the fourth invariance degree represent the most "harmless" kind of changes, because they cover changes in the body part of a component. The component interfaces are not modified by these changes.

Component based software development is the first important step towards building evolutionary software systems. But we must still be at an early stage with our attempts to build component based software systems. Discussions in academia and industry show that there is a huge gap in terminology, application, usage, side effects, and many other issues. The areas of compiler construction, databases, or functional programming provoke fewer discussions regarding their basic elements.

This session will discuss the application of component-based systems in distributed, real-time systems and techniques for developing connectors. A real challenge with state of the art techniques is that there are large numbers of platforms, not necessarily exotic ones, where neither the traditional middleware techniques e.g. CORBA nor component techniques as defined in industrial applications can be applied. A particular example for these kind of systems, are given by network embedded, real-time, wireless systems.