

## Reexamining the Role of Interactions in Software Architecture

Christine Hofmeister

Computer Science and Engineering Department, Lehigh University  
19 Memorial Drive West, Bethlehem, Pennsylvania 18015, USA  
Hofmeister@cse.lehigh.edu

**Abstract.** Describing and understanding the interactions among computation units in a software system remains one of the most interesting challenges of software architecture. In the old days, computation units were statements, and interaction among them consisted of the control flow between them. Structured programming and modern languages have made this quite tractable for developers to describe and understand. But object-orientation, concurrency, and component-based systems have pushed us into a realm where interactions between computation units (components) are often quite complex and difficult to understand.

Over the years numerous kinds of models have been used to describe interactions, including state charts, UML sequence diagrams, Petri nets, special-purpose languages, connectors, etc. Another less common, approach is to take advantage of the duality of application state and interaction, either using interaction state as a proxy for application state, or vice versa. With models the typical approach is to describe the allowable interactions of a component via an interaction protocol, then use these to determine whether components may be composed and perhaps to create a new model describing the composed interactions. However, component interactions, like many other component properties, can depend on the context or environment in which the component operates, and on the other components it is composed with. Describing these things in an interaction protocol and developing a corresponding composition semantics remains a problem.

Another issue has arisen with the use of multiple views to describe the software architecture. Interaction behavior is a key part of many types of views, which results in the need to reconcile the relationship between interaction protocols across views. A layered model of interaction protocols may be useful here. For example, the lowest layer could be RPC and HTTP interactions in an execution view, the middle layer be the interactions in a module view, and the top layer be the logical interactions in a conceptual view. Adaptation of interactions in cases of incompatibility, e.g. via packaging, is also important, but typically addresses the relationship between interaction protocols in the same layer rather than across layers.

Some recent work has moved away from a strictly static analysis of interaction protocols to include run-time checking of interactions. This greatly expands the kinds of things that can be described and checked in an interaction protocol, but does not address problems of composing interaction protocols nor of checking their consistency across protocol layers. Solving these problems may require a paradigm shift, from describing interaction protocols as properties of components to treating interactions as entities in their own right, with their own sets of properties.