Interface-based Design

Thomas A. Henzinger

University of California, Berkeley
and
EPF Lausanne, Switzerland

Interfaces play a central role in the component-based design of hardware and software systems. Good interface design is based on two principles. First, an interface should expose enough information about a component so to make it possible to predict if two or more components work together properly, by looking only at their interfaces. Second, an interface should not expose more information about a component than is required by the first principle. The technical realization of these two principles depends, of course, on our interpretation of what it means for two or more components to "work together properly." A simple interpretation is offered by typed programming languages: a component that implements a function and a component that calls that function are compatible if the function definition and the function call agree on the number, order, and types of the parameters. We present richer notions of interfaces, which expose in addition to type information, also temporal information about components. For example, the interface of a file server with the two methods `open-file` and `read-file` may stipulate that `read-file` must not be called before `open-file` has been called. Symmetrically, the interface of a client specifies the possible sequences of `open-file` and `read-file` calls during its execution, so that a compiler can check if the file server and the client fit together. Such *behavioral interfaces*, which expose temporal information about a component and at the same time impose temporal requirements on the environment of the component, can be specified naturally using an automaton-based language [2]. The notion of compatibility can be refined further, by exposing real-time information and resource-use information, leading to *timed* and *resource interfaces* [4, 1]. For instance, resource interfaces can be used to ensure that no two components simultaneously access a unique resource.

We formally capture the requirements on interfaces by axiomatizing *interface theories* [3]. For example, the axiom of "independent implementability" of interfaces stipulates that if A and B are compatible interfaces, and A' is a component that conforms to interface A, and B' is a component that conforms to interface B, then the composition $A'||B'$ of the two components conforms to the composite interface $A||B$. For selected interface formalisms, such as behavioral, timed, and resource interfaces, we show that they satisfy the axioms of interface theories, and we discuss the following algorithmic problems:

1. *Compatibility*: given two interfaces, are they compatible?
2. *Conformance*: given an interface and a component, does the component conform to the interface?
3. *Synthesis*: given a component, what is its interface?

In particular, we show that the compatibility checking of interfaces amounts to solving a game in which the interfaces and the unknown environment represent players. Furthermore, we show that the conformance relationship between a component and its interface must be a contravariant one, which treats inputs and outputs differently. This distinguishes interface conformance from most notions of refinement used in formal verification.

The work reported here is joint with Arindam Chakrabarti, Luca de Alfaro, and Marielle Stoelinga.

References

1. A. Chakrabarti, L. de Alfaro, T.A. Henzinger, and M. Stoelinga. Resource interfaces. In *EMSOFT 03: Embedded Software*, Lecture Notes in Computer Science 2855, pages 117–133. Springer-Verlag, 2003.
2. L. de Alfaro and T.A. Henzinger. Interface automata. In *Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering*, pages 109–120. ACM Press, 2001.
3. L. de Alfaro and T.A. Henzinger. Interface theories for component-based design. In *EMSOFT 01: Embedded Software*, Lecture Notes in Computer Science 2211, pages 148–165. Springer-Verlag, 2001.
4. L. de Alfaro, T.A. Henzinger, and M. Stoelinga. Timed interfaces. In *EMSOFT 02: Embedded Software*, Lecture Notes in Computer Science 2491, pages 108–122. Springer-Verlag, 2002.