A Logical Interface Description Language
for Components

F. Arbab\textsuperscript{1}, F.S. de Boer\textsuperscript{2}, and M.M. Bonsangue\textsuperscript{1}

\textsuperscript{1}CWI, Amsterdam, The Netherlands
{farhad,marcello}@cwi.nl
\textsuperscript{2}Utrecht University, The Netherlands
frankb@cs.uu.nl

Abstract. Motivated by our earlier work on the IWIM model and the Manifold language, in this paper, we attend to some of the basic issues in component-based software. We present a formal model for such systems, a formal-logic-based component interface description language that conveys the observable semantics of components, a formal system for deriving the semantics of a composite system out of the semantics of its constituent components, and the conditions under which this derivation system is sound and complete. Our main results in this paper are the theorems that formulate the notion of compositionality and the completeness of the derivation system that supports this property in a component-based system.

1 Introduction

Building applications out of software components is currently a major challenge for Software Engineering. The urgency and importance of this challenge are intensified by the continuous rapid growth of the supply and demand for software (components) on the internet, and the prospect of mobile computing. There are close ties between many of the issues investigated in the coordination research community in the past decade or so, on the one hand, and some of the basic problems in Component Based Software Engineering, on the other.

Motivated by our earlier work on the IWIM model and the Manifold language, in this paper we introduce a formal logic-based interface description language for components in component-based systems. We consider components as black box computational entities that communicate asynchronously via unbounded FIFO buffers. Each such FIFO buffer is called a channel and has a system-wide unique identity. The identity of a channel can also be communicated as a value through channels. This allows dynamic reconfiguration of channel connections among the components of a system.

The interface of a component describes its observable behavior abstracting away its implementation in a particular programming language. The interface of a component contains five elements: a name, a channel signature, and three predicates, namely a blocking invariant, a precondition, and a postcondition. The name of a component uniquely identifies the component within a system.
The channel signature of a component is a list of channels representing its initial connections. The blocking invariant is a predicate that specifies the possible deadlock behavior of the component. The precondition is a predicate that specifies the contents of the buffers of the initial external channels (i.e., the ones in the channel signature) of the component. The postcondition is a predicate that specifies the contents of the buffers of the external channels that exist upon termination.

In order to simplify our presentation in this paper, we restrict ourselves to component-based systems that consist of a static number of components and channels, although the connections in the system can change dynamically and in an arbitrary manner. Semantically, we describe the behavior of a component by a transition system, abstracting away from its internal details and the language of its implementation. We define the observable behavior of a component in terms of sequences of values, one for each channel-end that the component has been connected to. Thus, we abstract away the ordering among the communications on different channels. The observable behavior of a component-based system is given by the set of final global states of successfully terminating computations, provided that the system is deadlock-free. The existence of a deadlocking computation is considered a fatal error. A global state records for each channel the contents of its buffer.

The main contribution of this paper is to show that it is possible to reason about the correctness of an entire system compositionally in terms of the interface specifications of its components, abstracting away their internal implementation details. Our notion of correctness of a component-based system is based on the above-mentioned concept of observable behavior. This extends the usual notion of partial correctness by excluding deadlocks.

Compositionality is a highly desirable, but elusive, property for formal models of component-based systems. For compositionality to hold, the formal system that relates the semantics of the whole system to that of its individual components must constitute a proof method that is both sound and complete. We show that our proof method is generally sound. On the other hand, it is not generally possible to derive the formal semantics of a whole system as a composition of the local semantics of its components only. Consequently, completeness of our proof method does not generally hold. However, we show that it is possible to obtain completeness for component-based systems that satisfy certain restrictions. Indeed, we show that these restrictions are both necessary and sufficient conditions for completeness.

To achieve completeness, we impose two restrictions on component-based systems. First, we restrict to channels that are one-to-one and uni-directional. This means that every channel is an exclusively point-to-point communication medium between a single producer and a single consumer. The producer or the consumer of a channel loses its exclusive control of its channel-end by writing its identifier end to another channel. Subsequently, a component may dynamically (re)gain the exclusive control of a specific end of a channel, simply by reading its