A Fully Abstract Semantics for UML Components

F.S. de Boer¹,², M.M. Bonsangue²,⁎, M. Steffen³, and E. Ábrahám³

¹ CWI, Amsterdam, The Netherlands frb@cwi.nl
² LIACS, Leiden University, The Netherlands marcello@liacs.nl
³ Christian-Albrechts-University, Kiel, Germany {ms, eab}@informatik.uni-kiel.de

Abstract. We present a fully abstract semantics for components. This semantics is formalized in terms of a notion of trace for components, providing a description of the component externally observable behavior inspired by UML sequence diagrams. Such a description abstracts from the actual implementation given by UML state-machines. Our full abstraction result is based on a may testing semantics which involves a composition of components in terms of cross-border dynamic class instantiation through component interfaces.

1 Introduction

The Unified Modelling Language (UML)¹³ is widely adopted as the de facto industry standard for modelling object-oriented software systems. It consists of several graphical notations providing different views of the system being modelled. There are two basic types of diagrams: behavior diagrams and structure diagrams. These diagrams include sequence diagrams, state machines, class diagrams and component diagrams.

We use UML for investigating features such as state encapsulation, and name-passing in synchronous communication in combination with dynamic class instantiation. Basically, in UML a component is a set of classes with explicit contextual dependencies. Some instances of classes of a component are called ports. Components can communicate only through their ports. Most importantly, a port of a component can also instantiate new ports of another component. The explicit context dependencies of a component guarantee that ports have enough structural information about the environment. However the behavior of such an external environment is not under control of the component itself. In other words, a component is an open program, with implementation code containing calls to operations and constructors of interfaces that are not bound to any particular behavior specification.

From the point of view of a component, the ports of other components belong to the environment, and are internally known only as typed identifiers. Although the behavior of the environment is not fixed at priori, it has to obey to certain laws. For example, because the state of a port is encapsulated, external ports cannot always communicate

⁎ The research of Dr. Bonsangue has been made possible by a fellowship of the Royal Netherlands Academy of Arts and Sciences.
with each other. To illustrate this, consider a port of a component $i$ that creates two new ports $e_1$ and $e_2$ of some component in the environment. The ports $e_1$ and $e_2$ are both external, but unable to communicate with each other unless the internal object $i$ let one of them know the identity of the other. The above situation is characteristic of a framework with dynamic scope: new clusters of objects that know each other can be created as new external instances appear, and old clusters may merge as a consequence of a communication.

### 1.1 Contribution of This Paper

In this paper we select a subset of UML notations suitable as basis for modelling component-based systems. Inspired by UML sequence diagrams, we give a denotational semantics to UML components in terms of traces of their externally observable events. A trace describes a sequence of interactions between the ports of a set of components. Here a port is an instance of a class of a component realizing one of its interface, and an interaction is a synchronization on an operation declared on one of the interface of a component.

We define an observational equivalence for components based on may testing, and show that ordinary traces are, in general, not fully abstract: two components can be observationally equivalent but their associated set of traces be different. Our main result is the characterization of trace abstractions that takes into account the clustering structure of objects dictated by their dynamic scope. These traces are full abstract with respect to may testing observational equivalence.

### 1.2 Related Work

There is an increasing interest to give a rigorous foundation to UML for addressing, e.g., the needs for modelling safety critical applications. Some approaches are based on translating UML subsets into existing formalisms, like the $\pi$-calculus [19], other have proposed new meta-modelling language calculi as foundation for the semantics of UML, e.g. [11]. In this paper we present a variant of the UML subset considered by Damm et al. and formalized as a transition system [12]. The most significant departures from this work are that we do not consider asynchronous inter-object communications and do not distinguish among active, reactive and passive objects.

There are several full abstraction results for may-testing semantics for calculi of processes interacting in dynamically changing communication topology [6,14]. The UML description of classes by state-machines combines mechanisms for dynamic process creation similarly to object calculi [11,10,20,16] with synchronization mechanisms as in process calculi [9,6,14].

The closest work to our is Jeffrey and Rathke [16] fully abstract semantics of concurrent objects. While our components are open, programs in [16] are closed, in the sense we explained above, since their creation of a new object involves the specification of the behavior of the newly created object. Consequently, in their setting, the environment can be basically viewed as a static and a priori given group of objects. This contrasts with our setting, where the program itself creates dynamically its own environment and imposes constraints on the communication topology of its environment.