Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 176 (2007) 47–67
www.elsevier.com/locate/entcs

Extending a Component Specification Language with Time

Björn Metzler[1] and Heike Wehrheim[2]

Institut für Informatik
Universität Paderborn
33098 Paderborn, Germany

Abstract

In a formal approach to component specification, interfaces are usually described using pre- and postconditions of methods or protocols. In this paper we present an approach for integrating *time* into a component specification language which already allows for pre/post and protocol descriptions. The specification of timing aspects is indispensable when treating components of embedded systems underlying hard real-time requirements. In order to allow for a smooth integration into the existing specification language and to ease reading and writing of interfaces, we do not extend the language with yet another formalism (for time), but instead only add a specific feature (i.e. *clocks*) to it. We define a semantics for this new specification language in terms of *timed automata*, which thus also opens the possibility of analysing interface descriptions with the UPPAAL model checker. We furthermore give *timed simulation* conditions and prove their soundness with respect to inclusion of timed traces, the notion of implementation in timed automata. This implementation relation can be used as a correctness criterion for interoperability and substitutability checks.

Keywords: Interface specification, timed automata, pre/post conditions, protocols, simulation, verification.

1 Introduction

Interfaces of components are typically described by giving signature lists, pre- and postconditions of methods or by defining protocols (i.e. valid call sequences). Different approaches and languages have been proposed for these purposes: the signature list only technique is the approach adopted by most industrial middleware platforms, pre- and postconditions are for instance used in [16,28,18] and protocol definitions for components given as finite state automata, process algebra descriptions or temporal logic can be found in [20,21,11]. For embedded systems, it is however also

[1] Email: bmetzler@uni-paderborn.de
[2] Email: wehrheim@uni-paderborn.de

1571-0661
doi:10.1016/j.entcs.2006.02.031

important to specify *timing constraints* of interfaces, such as deadlines guaranteed or expected by a component.

In this paper we set out to develop a component specification language which allows for the specification of pre- and postconditions, protocols and timing constraints. The starting point here is an already existing language which contains features for specifying pre- and postconditions and protocols. The notation, called CSP-OZ [9], is a combination of the process algebra CSP [12,22] with the state-based, object-oriented formalism Object-Z [24]. The process algebra is used to specify specific call sequences guaranteed/expected by the component, the state-based formalism handles data-dependent aspects like pre- and postconditions of methods. Both formalisms come with built-in notions of *refinement* which is the formal development concept guaranteeing *substitutability*. Thus refinement can be used as correctness criterion for interoperability and substituability checks. The *integrated notation* CSP-OZ is now extended to allow for the specification of timing constraints. To this end, we however do not integrate a third formalism into the existing combination but instead only add a specific *clock* type for Object-Z variables. Clock variables can be declared, queried and changed just like ordinary variables. These clock variables allow for the specification of deadlines, minimum and maximum delays between method calls etc.. This is similar to the way finite automata are extended to timed automata [1], which is the standard formalism for describing systems with timing aspects (they, however, do not allow for a high level description of state-based and behavioural aspects).

For this specification language (called *timed* CSP-OZ) we furthermore propose a method for analysing component interfaces and we define a formal notion of implementation, which can - like refinement - be used for substitutability checks. The analysis method is based on a semantics for the language in terms of timed automata (or more precisely, timed transition systems, since the semantics will not always yield a finite state automaton). In case of a finite number of states we can then use one of the timed automata model checkers for verification (e.g. Kronos [27] or UPPAAL [3]).

Based on this semantics we can furthermore use the notion of implementation associated with timed automata for timed CSP-OZ. The implementation relation for timed automata is inclusion of *timed traces* (language inclusion for words with time stamps). We define *timed simulation* conditions and show their soundness with respect to this relation. This opens the way for a stepwise proof of implementation.

The paper is structured as follows. Next, we start with a simple example of a timed CSP-OZ specification on which we explain the general idea and which will serve as an illustration of the main results in the next sections. Section 3 gives a short introduction to timed automata. We then define the semantics for timed CSP-OZ specifications in terms of timed automata. In Section 4 we show how to analyse interface specifications in timed CSP-OZ with the timed automata model checker UPPAAL. Section 5 gives timed simulation conditions which can be used to prove