7 Component-Interaction Automata Approach (CoIn)

Barbora Zimmerova*, Pavlína Vařeková**, Nikola Beneš**, Ivana Černá***, Luboš Brim†, and Jiří Sochor**
Masaryk University, Brno, Czech Republic

7.1 Introduction

The aim of the CoIn approach (Component-Interaction Automata approach) is to create a framework for formal analysis of behavioural aspects of large scale component-based systems. For the modelling purpose, we use the Component-interaction automata language [1]. For the verification, we employ a parallel model-checker DiVinE [2], which is able to handle very large, hence more realistic, models of component-based systems. Verified properties, like consequences of service calls or fairness of communication, are expressed in an extended version of the Linear Temporal Logic CI-LTL.

7.1.1 Goals and Scope of the Component Model

The Component-interaction automata model behaviour of each component (both basic and composite) as a labelled transition system. The language builds on a simple, yet very powerful, composition operator that is able to reflect hierarchical assembly of the system and various communication strategies among components. Thanks to this operator, the language can be used for modelling the behaviour of components designed for or implemented in various component frameworks and models.

7.1.2 Modeled Cutout of CoCoME

While modelling the CoCoME, we have focused on the aspect of communicational behaviour of components. Hence we did not treat aspects like non-functional properties or data manipulation. However in terms of component interaction, we have modelled the CoCoME completely in fine detail, based on the provided Java implementation of the system. We modelled also parts like GUI, Event Channels, or the Product Dispatcher component. The final model was verified using the DiVinE verification tool. We have checked the compliance of the model to the Use Cases from Chapter 3, we have verified the Test Cases, and various other CI-LTL properties.

* Supported by the Czech Science Foundation within the project No. 102/05/H050.
** The authors have been supported by the grant No. 1ET400300504.
*** The author has been supported by the grant No. GACR 201/06/1338.
† The author has been supported by the grant No. 1ET408050503.
7.1.3 Benefit of the Modeling

One of our main benefits is that we use a general formal modelling language that is, thanks to its flexible composition operator, able to capture behaviour of various kinds of component-based systems. The created model is in fact a labelled transition system, which is a format that can be directly verified using a large variety of existing methods.

7.1.4 Effort and Lessons Learned

For the full modelling of the CoCoME, we needed two person months. The verification was then performed automatically using the DiVinE tool. This exercise helped us to discover the limits and capabilities of our modelling language and verification methods. In particular, we have found general solutions to various modelling issues, like creation and destruction of instances, modelling of global shared state, or exception handling. In terms of verification, we have examined the efficiency of our verification methods on a large model of a real system.

7.2 Component Model

Our framework is represented by the Component-interaction automata language. It should be emphasized that Component-interaction automata are not meant to support implementation of component-based systems. They are intended as a modelling language that can be used to create detailed models of behaviour of component-based systems to support their formal analysis.

The language is very general, which follows from two things. First, the Component-interaction automata language does not explicitly associate action names used in the labels of automata with interfaces/services/events/etc., which allows the designers to make the association themselves. The association must only respect that if the same action name is used in two components, in one as an input and in the other one as an output, it marks a point on which the components may communicate. Second, the language defines one flexible composition operator that can be parametrized to simulate several communication strategies used in various component models. In this manner, Component-interaction automata can be instantiated to a particular component model by fixing the composition operator and semantics of actions.

7.2.1 Definition of a Component-Interaction Automaton

Component-interaction automata capture each component as a labelled transition system with structured labels (to remember components which communicated on an action) and a hierarchy of component names (which represents the architectural structure of the component).

A hierarchy of component names is a tuple $H = (H_1, \ldots, H_n)$, $n \in \mathbb{N}$, of one of the following forms, $S_H$ denotes the set of component names corresponding to $H$. The first case is that $H_1, \ldots, H_n$ are pairwise different natural numbers; then