The CoIn Tool: Modelling and Verification of Interactions in Component-Based Systems

N. Beneš, L. Brim, I. Černá, J. Sochor, P. Vařeková and B. Zimmerova[1,2]

Faculty of Informatics, Masaryk University
Brno, Czech Republic

Abstract

The paper presents a modelling and model-checking tool designed for formal verification of interactions among components in hierarchical component-based systems. As distinct from existing verification frameworks, the presented *CoIn Tool* is able to analyse complex hierarchical models on the fly, and to verify linear temporal properties involving both state and action propositions.

Keywords: Component-based systems, Component-Interaction automata, formal verification, LTL, model checking.

1 Introduction and Motivation

Construction of correct component-based systems out of individual components is in principle a very difficult task, partly due to a high possibility of discrepancies in interaction of the components. Since the components are often developed independently of each other, the correctness of the cooperation logic among them becomes a significant issue.

One of the methods with the capacity to tackle this issue is the formal verification, and the LTL model checking [6,14] in particular. The existing LTL model-checking tools [1,10] are however not prepared to cope with a number of component-interaction specifics, like hierarchical structure of the models, with various compositional strategies applied at each level of the hierarchy, or incorporation of information about communicating counterparts. The component-oriented tools for component-interaction verification [11,13] often support verification of a limited set of predefined properties, or are restricted to either purely state-based or action-based reasoning about system properties. The combination of state-based and action-based reasoning is nevertheless very important in software verification [5].

[1] Email: {xbenes3,brim,cerna,sochor,xvareko1,zimmerova}@fi.muni.cz

[2] The work has been supported by the grants No. 1ET400300504 and No. 1ET408050503.

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

The *CoIn Tool* presented in this paper supports formal modelling of component interactions in a language designed for this purpose, and verification of a wide range of linear temporal properties on the model, combining both state-based and action-based reasoning.

2 The CoIn Tool

The CoIn Tool [7] consists of a *modelling platform*, which allows users to enter studied model in a graphical manner, and a *verification platform*, which verifies the model against a given property. The verification is fully automatic and proceeds on the fly, which helps to reduce the space requirements of the verification process, and hence allows the method to analyse large models. If the property is not valid on the model, the technique reports the behaviour of the model that violates the property.

2.1 Modelling Platform

The Modelling Formalism. The model of the studied system is created in the *Component-Interaction automata* language [4,15]. Component-Interaction automata (CI automata) are an automata-based formalism for behavioural modelling of components, i.e. interplay of their services and consequent calls of services provided by other components. The behaviour of each component is in the language represented with a labelled transition system (see Figure 1 on the left), which can be composed with automata of other components via synchronization on complementary labels [3], similarly to CCS.

During the modelling process, only the primitive components (on the lowest level of the system hierarchy) need to be modelled explicitly, defining their transition space, initial states and name. Composite components (and all their characteristics) are determined by the set of direct sub-components, and the composition type (see Figure 1 on the right). The composite components are kept in this compact form also during verification, when their transition space is generated and traversed on the fly.

While keeping the models finite-state, the CI-automata language is able to faithfully model component interaction in systems based on various component models, and to simulate some of the existing modelling formalisms designed for a similar purpose, e.g. [8,12]. The capacity of the formalism has been evaluated on the CoCoME case study [15], which includes also the guidelines for the usage of the language.

Implementation of the Modelling Platform. The modelling part of the CoIn Tool is a graphical Java application for visual modelling of component interaction with CI automata. In the tool, both primitive and composite components can be modelled and edited visually, which facilitates the creation of the model.

[3] The labels are triples which are classified according to their structure to input labels $(-, a, 1)$, saying that component 1 inputs action a; output labels $(2, a, -)$, saying that component 2 outputs action a; and internal labels $(2, a, 1)$ resulting from the synchronization of the two. Every input and output label represents an attempt for communication, whereas internal labels keep information about realized interactions.

The core functionality of the modelling platform includes visual entering of states and transitions of primitive-component models, including adjustment of their visual properties, automatically computed characteristics, like the hierarchy of component names, visual manipulation (zooming) and image export. The tool facilitates the description of composite-component models by supplying users with the lists of possible parameters and values, and syntactic checks during modelling.

The final model, structured into descriptions of primitive components and their compositions, is exported to one file in the CI-automata textual notation, which is passed to the verification platform.

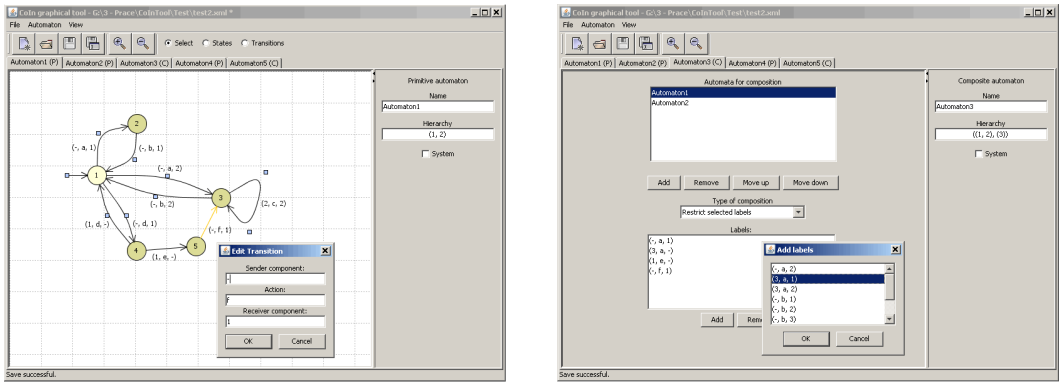

Fig. 1. Screenshots of the Modelling Platform

2.2 Verification Platform

The verification process has two inputs, the model of the studied system (in the CI-automata textual notation generated by the modelling platform) and the temporal property that is to be verified.

The Property-Specification Formalism. The properties can be specified in two temporal logics defined in [2]. The first is the *state/event LTL* (*SE-LTL* for short), which is a combination of the *state-based* an *action-based linear temporal logic (LTL)*, and hence includes both state-based and action-based LTL as its special cases. The second is the *weak SE-LTL*, which allows users to express such state/event properties about systems that automatically overlook uninteresting actions/interactions in system models, determined beforehand together with the property.

The difference between the two logics can be demonstrated already on the basic run proposition $\mathcal{P}(a)$, which in SE-LTL expresses that action *a Proceeds* as the first action of the run, whereas in weak SE-LTL it states that action *a Proceeds* as the first *interesting* action of the run. Thus when interpreted in weak SE-LTL, the action can be preceded by a number of uninteresting actions. This comes in useful namely when one aims to verify interactions among specific components, which can be interleaved with uninteresting behaviour of the rest of the system.

Notice that although LTL is a linear-time logic, the combination of the state-based and action-based LTL enables the properties to address also one-step branch-

ing of actions [3]. This is because the branching of transitions from a state can be encoded as a proposition of the state, namely via the set of actions enabled in the state. The *enabledness* propositions are currently the only state propositions we consider, since they do not require any explicit state-labelling of the model. However, the verification method is ready to handle general atomic propositions.

Implementation of the Verification Platform. The verification part of the CoIn Tool is realized as a C++ command-line application implementing the automata-based model checking [6,14], and consisting of two binaries. The first implements the *property conversion* and the second executes the *verification process*.

The property-conversion sub-tool converts a supplied temporal-formula property to an appropriate format (the property automaton) and appends it to the model of the system. The conversion is parametrized by the employed temporal logic, which is either SE-LTL (strong SE-LTL) or weak SE-LTL, and in case of weak SE-LTL also by a list of interesting action labels.

The verification sub-tool checks the model against the property, and reports a counterexample if the property is not valid on the model. Besides the verification, the tool can perform simple reachability of the model, reporting the number of reachable states, transitions and (possibly) deadlock states in the state space of the model. The reachable state space can be exported in a graphical form.

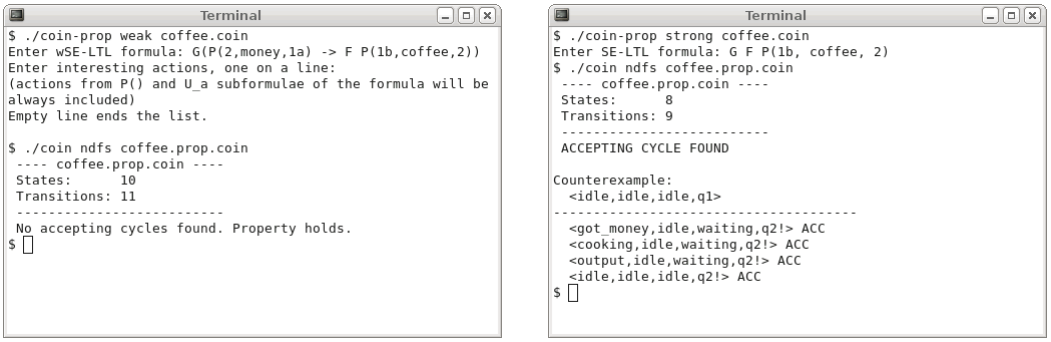

Fig. 2. Screenshots of the Verification Platform

3 Summary and Future Directions

There are two main aspects that distinguish the presented tool from related work. First, the model of the system is created with a formalism specially designed for component-interaction modelling, which allows the tool to analyse the core aspects of component interaction. Second, the tool is able to verify a wide range of temporal properties, combining both state-based and action-based reasoning, which is a significant advantage in connection with software verification [5].

Both the capability of the modelling language and application of a state/event logic to component-interaction verification have been evaluated on extensive case studies. In [15], we have presented a CI-automata model of a complex component-based system, designed as a common modelling example comprising a large number of various component-specific aspects. In [3], we have identified a set of proper-

ties reflecting common correctness issues in component-based systems, formalized them in terms of a state/event temporal logic, and demonstrated the feasibility and efficiency of their automatic verification.

The current release of the tool implements basic sequential model checking with no supplementary optimizations. We currently experiment with a version of the tool that implements distributed model checking [3], and we implement partial order reduction for the state/event logic [2]. Besides the release of these two extensions, we aim at developing a multi-core version of the tool, based on the DiVinE toolset [1,9].

Acknowledgments

We thank the DiVinE development team [9] for the property-to-automaton algorithm, and Milan Křivánek for implementing the modelling platform of the tool.

References

[1] Barnat, J., L. Brim, I. Černá, P. Moravec, P. Ročkai and P. Šimecek, *DiVinE – A Tool for Distributed Verification*, in: *Proceedings of the Computer Aided Verification conference (CAV'06)* (2006), pp. 278–281.

[2] Beneš, N., L. Brim, I. Černá, J. Sochor, P. Vařeková and B. Zimmerova, *Partial Order Reduction for State/Event LTL*, Technical Report FIMU-RS-2008-07, Masaryk University, Faculty of Informatics, Brno, Czech Republic (2008).

[3] Beneš, N., I. Černá, J. Sochor, P. Vařeková and B. Zimmerova, *A Case Study in Parallel Verification of Component-Based Systems*, in: *Proceedings of the Workshop on Parallel and Distributed Methods in verifiCation (PDMC'08)*, ENTCS (2008), pp. 35–51.

[4] Brim, L., I. Černá, P. Vařeková and B. Zimmerova, *Component-Interaction Automata as a Verification-Oriented Component-Based System Specification*, in: *Proceedings of the ESEC/FSE Conference on Specification and Verification of Component-Based Systems (SAVCBS'05)* (2005), pp. 31–38.

[5] Chaki, S., E. M. Clarke, J. Ouaknine, N. Sharygina and N. Sinha, *State/Event-Based Software Model Checking*, in: *Proceedings of the International Conference on Integrated Formal Methods (IFM'04)*, LNCS **2999** (2004), pp. 128–147.

[6] Clarke, E. M., O. Grumberg and D. A. Peled, "Model Checking," The MIT Press, USA, 2000, ISBN 0-262-03270-8.

[7] *CoIn Tool – Formal-Verification Tool of Component-Interaction.*
URL http://anna.fi.muni.cz/coin/tool

[8] de Alfaro, L. and T. A. Henzinger, *Interface-based Design*, in: *Proceedings of the 2004 Marktoberdorf Summer School* (2005), pp. 1–25.

[9] *DiVinE – Distributed Verification Environment.*
URL http://anna.fi.muni.cz/divine

[10] Holzmann, G. J., *The Model Checker SPIN*, IEEE Transactions on Software Engineering **23** (1997), pp. 279–295.

[11] Mach, M., F. Plášil and J. Kofroň, *Behavior Protocols Verification: Fighting State Explosion*, International Journal of Computer and Information Science **6** (2005), pp. 22–30.

[12] Plášil, F. and S. Višňovský, *Behavior Protocols for Software Components*, IEEE Transactions on Software Engineering **28** (2002), pp. 1056–1076.

[13] Plšek, A. and J. Adámek, *Carmen: Software Component Model Checker*, in: *Proceedings of the International Conference on the Quality of Software-Architectures (QoSA'08)* (2008).

[14] Vardi, M. Y. and P. Wolper, *An Automata-Theoretic Approach to Automatic Program Verification*, in: *Proceedings of the IEEE Symposium on Logic in Computer Science (LICS'86)* (1986), pp. 332–344.

[15] Zimmerova, B., P. Vařeková, N. Beneš, I. Černá, L. Brim and J. Sochor, "The Common Component Modeling Example: Comparing Software Component Models," LNCS **5153**, Springer-Verlag, 2008 pp. 146–176.