Interaction Correctness of Component-Based Systems

Barbora Zimmerova

Faculty of Informatics, Masaryk University
Brno, Czech Republic

Co-authors: I. Černá, J. Sochor, L. Brim, P. Vařeková, and N. Beneš

November 3, 2006
1 Introduction
   Team introduction
   Motivation and objectives of our work

2 Specification language
   Component-interaction automata

3 Preliminary results
   Verification of coordination errors
   Reconfiguration correctness

4 Conclusion
   Summary of the talk
Team introduction

Domain: Hierarchical component-based software systems
- Component interactions in particular
- Verification and formal analysis of component-based systems

Team:
- Verification of Component-Based Systems Group
  ParaDiSe Laboratory, FI MU, Brno, Czech Republic
  www.fi.muni.cz/paradise
- Ivana Černá, Jiří Sochor, Luboš Brim, Barbora Zimmerova, Pavlína Vařeková, and Nikola Beneš
Component-based systems

Benefits of

- reusing prefabricated components
- increasing flexibility of the software product
- reducing development costs

Serious questions on correctness of

- each primitive component
  - third party certification
- the assembled system
  - interconnection logic of component interactions
Issues of our interest

• **Verification of coordination errors**
  - Deadlock, computational progress, ...
  - Interaction between specific components

• **Reconfiguration correctness**
  - Component substitutability
  - Regression verification

• **Assembly strategies**
  - Identification of proper components
  - Binding of the components together

• **Component-interaction analysis**
  - Removal of inactive components
  - Component placement in distributed environment
1 Introduction
   Team introduction
   Motivation and objectives of our work

2 Specification language
   Component-interaction automata

3 Preliminary results
   Verification of coordination errors
   Reconfiguration correctness

4 Conclusion
   Summary of the talk
Component-interaction automata language

Component-Interaction automata language
(or CI automata for short)

- Automata-based language
  finite state model, infinite executions/traces

- Three types of actions (*input*, *output* and *internal*)
  no additional semantics – interfaces/services/provisions/requirements/
  requests/responses/events

- One-to-one synchronization
  one input and one output action which becomes internal later on

- Captures important interaction information
  participants of communication, hierarchy

- Flexible composition
  can be parametrized by architectural assembly, communication strategy
Definition of a CI automaton

A component-Interaction automaton

- States (initial)
- Labeled transitions
- Labels (structured - component names, actions)
  - input, output and internal
- Hierarchy

\[
\begin{align*}
C_1 : & \quad p \xrightarrow{(1,c,-)} r \quad \text{Hierarchy: (1)} \\
C_2 : & \quad p \xrightarrow{(2,a,-)} q \quad \text{Hierarchy: (2)} \\
C_3 : & \quad p \xrightarrow{(3,d,4)} q \quad \text{Hierarchy: ((3),(4))}
\end{align*}
\]
Composition of CI automata

A parametrizable composition operator \( \otimes^F \) determines a composite automaton \( \otimes^F S \) as

- a product of automata from \( S \)
- one-to-one synchronization
- the transitions with labels outside \( F \) are removed

**Application:**
The composition respects bindings between components given by feasible labels in \( F \).
Composition of CI automata - example

Example: The right blue figure depicts the composite automaton \( C = \otimes \mathcal{F} \{ C_1, C_2 \} \) where \( \mathcal{F} = \{ (2, a, 1), (1, b, 1), (1, c, 2) \} \).

Hierarchy: (1)

Hierarchy: (2)

Hierarchy: ((1),(2))
Introduction
Team introduction
Motivation and objectives of our work

Specification language
Component-interaction automata

Preliminary results
Verification of coordination errors
Reconfiguration correctness

Conclusion
Summary of the talk
Verification of coordination errors

Formal verification
- Behaviour of a system – (infinite) paths of a CI automaton
- Temporal properties
- Model checking

Model checking
- LTL-like logic for specification of properties
- Distributed-verification tool DiVinE
1 Introduction
   Team introduction
   Motivation and objectives of our work

2 Specification language
   Component-interaction automata

3 Preliminary results
   Verification of coordination errors
   Reconfiguration correctness

4 Conclusion
   Summary of the talk
Reconfiguration correctness

The issue:

- Component-based systems – evolve simply by component update
- Third party components – interaction correctness of updated system

Our solution:
Formal characterization of relationship between components which guarantees that the update does not break existing functionality of the system.
Reconfiguration correctness – Solution

Three specific issues:

- Substitutability of equivalent components
  via equivalence relation

- Substitutability of non-equivalent components
  via substitutability relation

- Independent implementability
  via specification–implementation relation, called compliance
1 Introduction
   Team introduction
   Motivation and objectives of our work

2 Specification language
   Component-interaction automata

3 Preliminary results
   Verification of coordination errors
   Reconfiguration correctness

4 Conclusion
   Summary of the talk
Summary of the talk

- **Introduction of our team**

- **Objectives of our research**
  - Verification of coordination errors
  - Reconfiguration correctness
  - Assembly strategies
  - Component-interaction analysis

- **Specification language**
  - Component-Interaction automata

- **Preliminary results**
  - Verification of coordination errors
  - Reconfiguration correctness
Thank you for your attention