Component-Interaction Automata

Barbora Zimmerova

Faculty of Informatics, Masaryk University, Brno
zimmerova@fi.muni.cz

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

January 18, 2006
1 Motivation
   Introduction
   Objectives

2 Related work
   Specification languages

3 Component-Interaction automata
   CI automata language
   Definition of the CI automaton
   Composition of CI automata

4 Example
   Simple system

5 Conclusion
   Summary of the talk
   Ongoing and future work
Introduction

**Domain:** Hierarchical component-based software systems

- Specification of interaction among components
- Verification of interaction properties

**Project:**

- Verification of Component-Based Systems
  FI MU, Brno, ParaDiSe Laboratory
  www.fi.muni.cz/paradise

- Ivana Černá, Jiří Sochor, Luboš Brim, Barbora Zimmerova, Pavlína Vařeková, and Nikola Beneš
Objectives – Verification

Issues of our interest

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

- Reconfiguration correctness
  - Component substitutability
  - Regression verification

- Component-interaction analysis
  - Removal of inactive components
  - Component placement in distributed environment
Objectives – Language

Language for specification of component interactions

1. Flexible – respect various component models
   - Respect the architecture of a system
   - Single/multiple bindings on interfaces
   - Synchronization strategies

2. Capture important information
   - Components – participants of communication
   - Hierarchical structure

3. Produce models of manageable size
   - To enable automated verification
Architecture description languages – languages that have been defined within frameworks of architecture description languages

- Tracta
- Wright
- SOFA Behavior protocols

Automata based languages – languages for specification of component interactions that have been defined independently

- I/O automata
- Team automata
- Interface automata
Specification languages – Overview

**Architecture description languages**

- **Tracta**
  - LTS, composition via multi-way synchronization, model checking
- **Wright**
- **SOFA Behavior protocols**

**Automata based languages**

- **I/O automata**
- **Team automata**
- **Interface automata**
Architecture description languages

- Tracta
- Wright
  CSP, multi-way synchronization, consistency and completeness checks
- SOFA Behavior protocols

Automata based languages

- I/O automata
- Team automata
- Interface automata
Specification languages – Overview

Architecture description languages

- Tracta
- Wright
- SOFA Behavior protocols
  regular-like expressions, one-to-one synchronization, compliance checking

Automata based languages

- I/O automata
- Team automata
- Interface automata
Specification languages – Overview

Architecture description languages

- Tracta
- Wright
- SOFA Behavior protocols

Automata based languages

- I/O automata
  - LTS, one-to-many synchronization, model checking and theorem proving
- Team automata
- Interface automata
Specification languages – Overview

Architecture description languages

- Tracta
- Wright
- SOFA Behavior protocols

Automata based languages

- I/O automata
- Team automata
  - LTS, flexible composition, security analysis
- Interface automata
Architecture description languages

- Tracta
- Wright
- SOFA Behavior protocols

Automata based languages

- I/O automata
- Team automata
- Interface automata
  LTS, one-to-one synchronization, compatibility/refinement checking
Specification languages – Discussion

1. Flexible – respect various component models
2. Capture important information
3. Produce models of manageable size

Architecture description languages
- Tracta
- Wright
- SOFA Behavior protocols

Automata based languages
- I/O automata
- Team automata
- Interface automata
Component-Interaction automata language

(CI automata for short)

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

- Three types of actions (*input*, *output* and *internal*)
  general used concept

- CCS like synchronization
  one input and one output action which becomes internal later on

- Flexible composition
  can be parametrized by characteristics of a system

- Preservation of important interaction information
  participants of communication, hierarchy
Definition of the CI automaton

Component-Interaction automaton

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

Hierarchy: (1)

\[ C_1 : \]

\[ C_2 : \]

Hierarchy: (2)

\[ (-, \text{Ins}, 1) \]

\[ (1, \text{Log}, 1) \]

\[ (1, \text{Done}, -) \]

\[ (2, \text{Ins}, -) \]

\[ (-, \text{Done}, 2) \]
Composition of CI automata

1. Set $S$ of CI automata to be composed
2. All possible interactions – complete transition space $\Delta_S$
3. CI automaton over $S$ – transition set $\delta \subseteq \Delta_S$
   (may be determined by architecture and other characteristics of the system)

Hierarchy: (1) $C_1 : 0 \rightarrow 1 \rightarrow 2$

$C_1 : (1, Done, -) \\
(1, Log, 1) \\
(-, Ins, 1)

Hierarchy: (2) $C_2 : 0 \rightarrow 1\rightarrow 2$

$C_2 : (-, Done, 2) \\
(2, Ins, -)
Composition of CI automata

\[ C_1 : \begin{array}{c}
0 \\
1 \\
2
\end{array} \]
\[ C_2 : \begin{array}{c}
0 \\
1
\end{array} \]

Hierarchy: (1)

Hierarchy: (2)

\[ Hierarchy: ((1),(2)) \]

In figures, states \( ij \) stand for \((i,j)\)
Simple system

- Consist of three components – database $C_1$, and clients $C_2$, $C_3$
- Respect architectural description
- Use one-to-one handshake synchronization

\[\text{Hierarchy: (1)}\]

\[\text{Hierarchy: (2)}\]

\[\text{Hierarchy: (3)}\]
Example – Simple system
Example – Simple system

Hierarchy: ((1),(2),(3))
Example – Simple system
Example – Simple system

Hierarchy: ((1),(2),(3))
Example – Simple system

Barbora Zimmerova

Component-Interaction Automata

January 18, 2006 23 / 27
Example – Simple system

Hierarchy: ((1),(2),(3))
Summary of the talk

**Component-Interaction automata** specification language for precise capturing of component interactions

1. **Flexible** – respect various component models
   flexible composition

2. **Capture important information**
   structured labels, hierarchy

3. **Produce models of manageable size**
   finite state LTS
Ongoing and future work

Ongoing work

- Behavioural equivalencies
- Composition operator (and others)
  - with respect to the architecture
  - with respect to synchronization
- Temporal logic

Future work

- Verification of coordination errors
- Reconfiguration correctness
- Component-interaction analysis
Thank you for your attention