CoCoME: Component-Interaction Automata Approach

The CoIn Team

B. Zimmerová, P. Vařeková, N. Beneš, I. Černá, L. Brim, and J. Sochor

Faculty of Informatics, Masaryk University
Brno, Czech Republic

August 1, 2007
1 Introduction

2 Component Model
   Component-interaction automata

3 Modelling the CoCoME
   Modelling technique
   Modelling overview
   Selected issues

4 Analysis
   Temporal-logic properties
   Use cases and test cases

5 Conclusion
1 Introduction

2 Component Model
   Component-interaction automata

3 Modelling the CoCoME
   Modelling technique
   Modelling overview
   Selected issues

4 Analysis
   Temporal-logic properties
   Use cases and test cases

5 Conclusion
Affiliation:
ParaDiSe Laboratory, Faculty of Informatics
Masaryk University, Brno, Czech Republic
http://anna.fi.muni.cz/coin

Members:

- **Profs:** Ivana Černá, Luboš Brim, Jiří Sochor
- **Studs:** Barbora Zimmerova, Pavlína Vařeková, Nikola Beneš

Experience:

- **ParaDiSe Laboratory (1999)**
  automated verification of large-scale systems
  verification tool DiVinE

- **CoIn Team (2005)**
  communication behaviour in component-based systems
1 Introduction

2 **Component Model**
   Component-interaction automata

3 Modelling the CoCoME
   Modelling technique
   Modelling overview
   Selected issues

4 Analysis
   Temporal-logic properties
   Use cases and test cases

5 Conclusion
Focus of our modelling approach

- Behavioural view
- Interaction among components

The purpose of the model

- Formal verification of component interaction

Framework represented by

- Component-interaction automata language
- For detailed modelling of communication behaviour in CBSs
- Very general → can be used with various component models
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/events/etc.

- Captures important interaction information
  participants of communication, hierarchy of components

- Flexible composition
  can be parametrized by architectural assembly, communication strategy

- General to meet various component models
  by fixing the composition operator and semantics of actions
Definition of a CI automaton

A component-Interaction automaton

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

Hierarchy: (2)

Hierarchy: (1)

Hierarchy: ((3),(4))

\[ C_1 : \rightarrow p \rightarrow q \rightarrow r \rightarrow \]
\[ (−, a, 1) \rightarrow (1, b, 1) \rightarrow (1, c, −) \rightarrow \]

\[ C_2 : \rightarrow p \rightarrow q \rightarrow \]
\[ (2, a, −) \rightarrow (−, c, 2) \rightarrow (3, a, 4) \rightarrow (4, c, 3) \rightarrow \]

\[ C_3 : \rightarrow p \rightarrow q \rightarrow \]

Hierarchy: (1)
Composition of CI automata

A parameterizable composition operator $\otimes_T$ determines a composite automaton $\otimes_T S$ as

- a product of automata from the set $S$
  $\rightarrow$ complete transition space $\Delta_S$
- where the transitions outside $T$ are removed
  $\rightarrow$ $T$ can reflect various communicational strategies
- composed hierarchically $\otimes_T \{ C_1, C_2 = \otimes_{T'} \{ C_4, C_5, C_6 \}, C_3 = \otimes_{T''} \{ C_7 \} \}$
  $\rightarrow$ the transition space determined by the expression, not computed explicitly!

Hierarchy: (1)

\[ C_1 : \quad \begin{array}{c}
  p \quad (1,c,-) \quad r \\
  \quad \downarrow \\
  \quad \uparrow \\
  \quad q \quad (1,b,1) \\
  \quad (-,a,1) \\
\end{array} \]

Hierarchy: (2)

\[ C_2 : \quad \begin{array}{c}
  p \quad (2,a,-) \quad q \\
  \quad \downarrow \\
  \quad \uparrow \\
  \quad r \quad (-,c,2) \\
\end{array} \]
Composition – complete transition space

Hierarchy: (1)

$C_1: \quad p \xrightarrow{(1,c,-)} r \xrightarrow{(1,b,1)} q$

Hierarchy: (2)

$C_2: \quad p \xrightarrow{(-,c,2)} (2,a,-) \xrightarrow{(-,c,2)} q$

$\Delta\{C_1,C_2\}$:

- $(p,p) \xrightarrow{(1,c,-)} (p,q) \xrightarrow{(-,a,1)} (q,p) \xrightarrow{(1,b,1)} (r,p)$
- $\cdots$
Composition – cube-like composition

\[ C = \otimes_T \{ C_1, C_2 \} \text{ where } T = \Delta_{\{C_1, C_2\}} \]

Hierarchy: ((1),(2))
Composition – handshake-like composition

\[ C = \bigotimes_T \{ C_1, C_2 \} \text{ where } T = \{(s, x, s') \mid x \in \{(2, a, 1), (1, b, 1), (1, c, 2)\}\} \]

Hierarchy: \((1)\)

Hierarchy: \((2)\)

Hierarchies: \(((1),(2))\)
1 Introduction

2 Component Model
   Component-interaction automata

3 Modelling the CoCoME
   Modelling technique
   Modelling overview
   Selected issues

4 Analysis
   Temporal-logic properties
   Use cases and test cases

5 Conclusion
Input

- **Specification of behaviour of primitive components**
  - Java implementation
- **Static structure of the system**
  - hierarchy of components, interfaces and bindings in between
  - derived from component diagrams and Java implementation

Output

- CI automaton representing the whole system
Modelling process

- Identify primitive components and their services
- Model primitive components as automata
  - An automaton for a service
  - An automaton for a primitive component via composition of the services
- Model composite components as automata
  - Fix the composition operator
  - An automaton for a composite component
- Proceed to formal analysis and verification
Modelling process

- Identify primitive components and their services
- Model primitive components as automata
  - An automaton for a service
  - An automaton for a primitive component via composition of the services
- Model composite components as automata
  - Fix the composition operator
  - An automaton for a composite component
- Proceed to formal analysis and verification
An automaton for a service

Each service, say `doIt()`, assigned a **tuple of actions**

- **call** of the method – `doIt`
- **return** from the method – `doIt'`

and modelled as a **loop**

\[
C_1: \quad (p, p) \xrightarrow{(\neg, doA, 1)} (q, q) \quad (1, doA', -) \\
\quad (q, p) \xrightarrow{(1, doB, -)} (r, r) \quad (1, doB', -) \\
\quad (r, p) \xrightarrow{(-, doB', 1)} (s, s) \quad (-, doB', 1)
\]

Hierarchy: (1)

\[
C_2: \quad (p, p) \xrightarrow{(-, doB, 2)} (q, q) \quad (2, doB', -) \\
\quad (q, p) \xrightarrow{(2, doC, -)} (r, r) \quad (2, doC', -) \\
\quad (r, p) \xrightarrow{(-, doC', 2)} (s, s) \quad (-, doC', 2)
\]

Hierarchy: (2)

\[
C = \bigotimes_T \{C_1, C_2\}: \quad (p, p) \xrightarrow{(-, doA, 1)} (q, p) \xrightarrow{(1, doB, 2)} (r, q) \\
\quad (q, p) \xrightarrow{(1, doA', -)} (s, p) \xrightarrow{(2, doB', 1)} (r, s) \xrightarrow{(2, doC', -)} (r, r) \\
\quad (r, q) \xrightarrow{(-, doC', 2)} (r, r)
\]

Hierarchy: ((1),(2))
An automaton for a primitive (basic) component

- Composition of automata for services using the star-like $C_4$ and the cube-like $C_3$ composition

Hierarchy: (1) $H_1$ $\rightarrow$ $H_2$ $\rightarrow$ (2)

Hierarchy: (1) $H_1$ $\rightarrow$ $H_2$ $\rightarrow$ (2)

Hierarchy: ((1),(2)) $H_1$ $\rightarrow$ $H_2$ $\rightarrow$ (1) $\rightarrow$ (2)
An automaton for a composite component

- Composition of automata for components using the handshake-like or the assembly-like composition

Assembly-like composition

- $T$ given explicitly as a set of transitions with labels representing interaction allowed by bindings among components

Example:

$$L_A = \{(−, sA, 1), (1, sA', −), (−, sC, 1), (1, sC', −), (1, sB, −), (−, sB', 1), (1, intA, 1)\}$$

$$L_B = \{(−, sB, 2), (2, sB', −), (2, sD, −), (−, sD', 2), (2, intB, 2)\}$$

$$F = \{(1, intA, 1), (2, intB, 2), (1, sB, 2), (2, sB', 1), (−, sC, 1), (1, sC', −), (2, sD, −), (−, sD', 2)\}$$

$$C_C = \bigotimes_T \{C_A, C_B\} \text{ where } T = \{(q, x, q') \mid x \in F\}$$
1 Introduction

2 Component Model
   Component-interaction automata

3 Modelling the CoCoME
   Modelling technique
   Modelling overview
   Selected issues

4 Analysis
   Temporal-logic properties
   Use cases and test cases

5 Conclusion
• **The whole Trading System** modelled in a fine detail
| 1 | Introduction        |
| 2 | Component Model    |
|    | Component-interaction automata |
| 3 | Modelling the CoCoME |
|    | Modelling technique |
|    | Modelling overview |
|    | Selected issues    |
| 4 | Analysis           |
|    | Temporal-logic properties |
|    | Use cases and test cases |
| 5 | Conclusion         |
Exception handling
- try, catch, finally blocks
- throw, delegate an exception

Creation and destruction of instances
- initial activation part
Internal state of a component

- additional automaton representing the internal state
- answers questions if (currState.equals(PAYING_BY_CASH))
  and reacts to commands currState = PAID;

Asynchronous messaging

- publish-subscribe communicational model
- realized via event channels
1 Introduction

2 Component Model
   Component-interaction automata

3 Modelling the CoCoME
   Modelling technique
   Modelling overview
   Selected issues

4 Analysis
   Temporal-logic properties
   Use cases and test cases

5 Conclusion
Input for the analysis

- Model of the system as a **Cl automaton**
- Labelled transition system (LTS) in fact

Analytical methods

- Variety of methods available for LTSs
- Verification of temporal properties with **Model Checking**
- **DiVinE tool** for verification of large-scale systems
- Application
  - In **design phase** to predict properties of a new system
  - Analysis and verification of **existing system**
  - During **modelling** to detect modelling errors
Logic for expressing properties CI-LTL

- Extended version of LTL, operators $\text{next } X$ and $\text{until } U$
- Both state and action-based
- Properties about
  - component interaction that is proceeding $P$
  - possible interaction that is enabled $E$

Verification

- DiVinE tool
  - distributed and on-the-fly model checking and reachability analysis
- Verification run on a cluster of 20 computers
- Presented properties verified in terms of seconds or minutes
Example of properties

- If the *StoreApplication (610)* starts a transaction with the *Persistence (511)*, it correctly closes the transaction before it is able to start another one.

  \[ G ( \mathcal{P}(610, \text{beginTransaction}, 511) \Rightarrow \mathcal{X} (\neg \mathcal{E}(610, \text{beginTransaction}, 511) \cup \mathcal{P}(610, \text{close}, 511))) \]

- It cannot happen that the *StoreApplication (610)* is ready to call *queryStockItemById()* but never can do so because its counterpart is never ready to receive the call.

  \[ G (\mathcal{E}(610, \text{queryStockItemById}, \neg) \Rightarrow \mathcal{F} \mathcal{E}(610, \text{queryStockItemById}, 521)) \]
1 Introduction

2 Component Model
   Component-interaction automata

3 Modelling the CoCoME
   Modelling technique
   Modelling overview
   Selected issues

4 Analysis
   Temporal-logic properties
   Use cases and test cases

5 Conclusion
Use cases

Application

- To check the model against the use case scenarios
- To find a path in the model that realizes the scenario
- To refine the scenario according to the path

All use cases confirmed using DiVinE

<table>
<thead>
<tr>
<th>Use Case</th>
<th>States</th>
<th>Transitions</th>
<th>Confirmed after generating</th>
</tr>
</thead>
<tbody>
<tr>
<td>UC 1: ProcessSale</td>
<td>401</td>
<td>1.488</td>
<td>18 of 384 states</td>
</tr>
<tr>
<td>(i)</td>
<td>10.600.010</td>
<td>63.819.991</td>
<td>85 of 3.965.100 states</td>
</tr>
<tr>
<td>(ii)</td>
<td>4.975.487</td>
<td>29.648.100</td>
<td>1.658.496 of 3.317.012 states</td>
</tr>
<tr>
<td>(iii)</td>
<td>181</td>
<td>211</td>
<td>487 of 876 states</td>
</tr>
<tr>
<td>UC 3: OrderProducts</td>
<td>181</td>
<td>211</td>
<td>487 of 876 states</td>
</tr>
<tr>
<td>UC 5: ShowStockReports</td>
<td>57</td>
<td>64</td>
<td>63 of 94 states</td>
</tr>
<tr>
<td>UC 7: ChangePrice</td>
<td>82</td>
<td>94</td>
<td>49 of 114 states</td>
</tr>
</tbody>
</table>

(i) UC 1: ProcessSale: CashPayment: btnStartNewSale
(ii) UC 1: ProcessSale: CashPayment: btnClose
(iii) UC 1: ProcessSale: CardPayment: btnEnterPIN
Test cases

Application

- To evaluate the test scenarios on the model

Informal scenarios

- Check for existence of a good behaviour
- Formulate a CI-LTL formula
- Verify in a negative way

Formal scenarios

- Check if all behaviours are good in some sense
- Formulate a CI-LTL formula and verify
- Consider only fair runs
1 Introduction

2 Component Model
   Component-interaction automata

3 Modelling the CoCoME
   Modelling technique
   Modelling overview
   Selected issues

4 Analysis
   Temporal-logic properties
   Use cases and test cases

5 Conclusion
Summary and lessons learned

Summary

- Application of Component-interaction automata to CoCoME
  - mapping of actions, composition operators, modelling process
- Solutions to various modelling issues
- Detailed automatic verification

Lessons learned

- The modelling language
  + high modelling capability
  − requires a lot of effort → current works on modelling support
- The verification techniques
  + verification of very large models
  + fully automatic
Thank you for your attention