

# CoCoME: Component-Interaction Automata Approach

The Coln Team

B. Zimmerova, 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

# Group introduction

## 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
- Coln 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

# Component Model

## 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

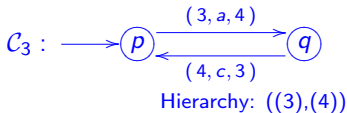
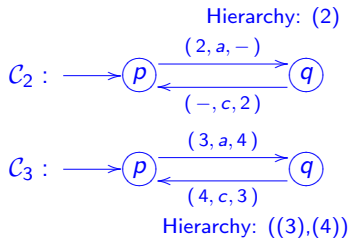
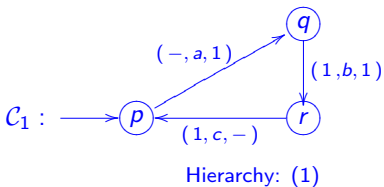
(or CI automata for short)

- 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

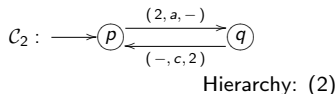
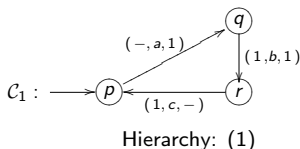




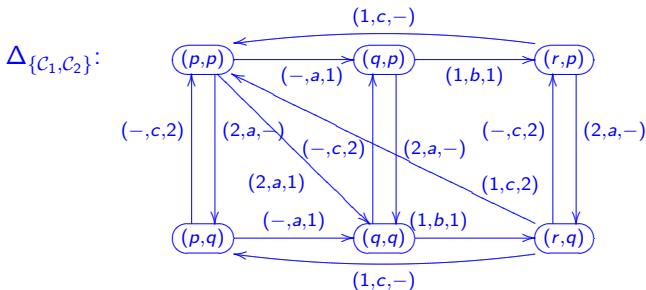
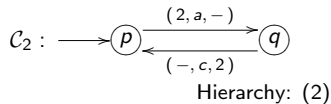
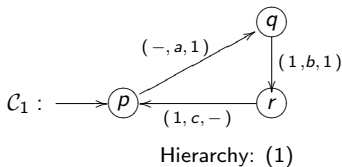
# Composition of CI automata

A parameterizable **composition operator**  $\otimes_T$  determines a **composite automaton**  $\otimes_T \mathcal{S}$  as

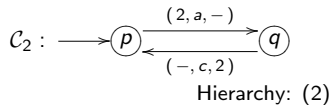
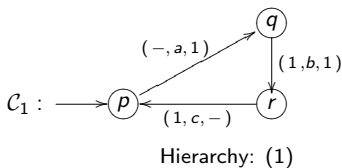
- a product of automata from the set  $\mathcal{S}$ 
  - **complete transition space**  $\Delta_{\mathcal{S}}$
- where the transitions outside  $T$  are **removed**
  - $T$  can reflect various **communicational strategies**
- composed **hierarchically**  $\otimes_T \{C_1, C_2 = \otimes_{T'} \{C_4, C_5,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



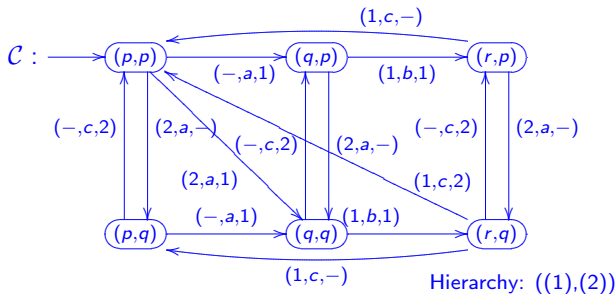
# Composition – complete transition space



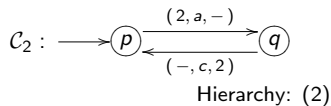
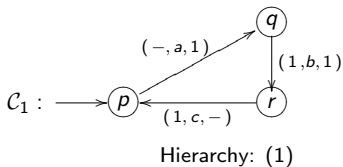
# Composition – cube-like composition



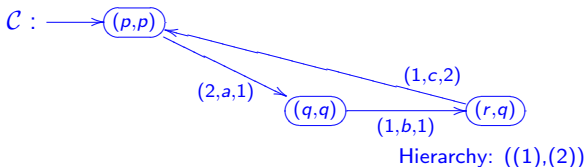
$$\mathcal{C} = \otimes_T \{\mathcal{C}_1, \mathcal{C}_2\} \text{ where } T = \Delta_{\{c_1, c_2\}}$$



# Composition – handshake-like composition



$$\mathcal{C} = \otimes_T \{\mathcal{C}_1, \mathcal{C}_2\} \text{ where } T = \{(s, x, s') \mid x \in \{(2, a, 1), (1, b, 1), (1, c, 2)\}\}$$



## 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

# Modelling technique

1/2

## 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

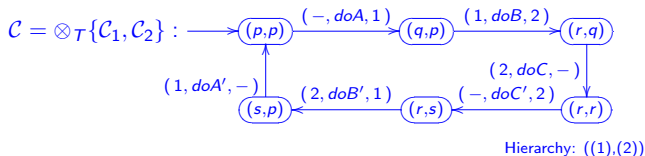
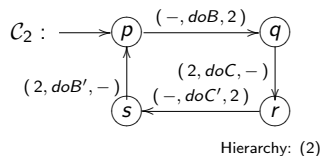
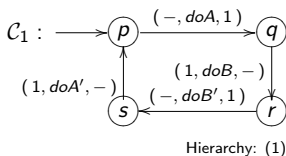


# An automaton for a service

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

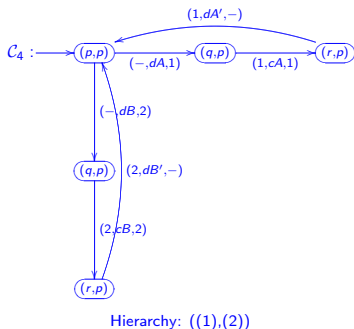
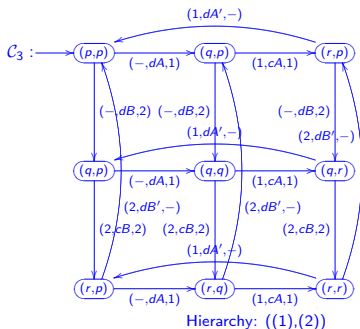
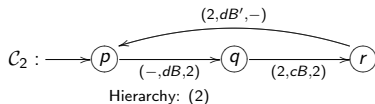
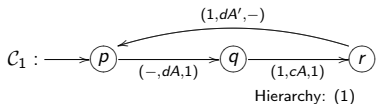
- **call** of the method – `doIt`
- **return** from the method – `doIt'`

and modelled as a **loop**



# 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



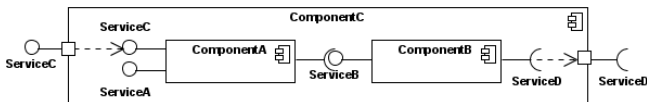
# An automaton for a composite component

- 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:*



$$\mathcal{L}_A = \{(-, sA, 1), (1, sA', -), (-, sC, 1), (1, sC', -), (1, sB, -), (-, sB', 1), (1, intA, 1)\}$$

$$\mathcal{L}_B = \{(-, sB, 2), (2, sB', -), (2, sD, -), (-, sD', 2), (2, intB, 2)\}$$

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

$$\mathcal{C}_C = \otimes_T \{\mathcal{C}_A, \mathcal{C}_B\} \text{ where } T = \{(q, x, q') \mid x \in \mathcal{F}\}$$

## ① Introduction

## ② Component Model

Component-interaction automata

## ③ Modelling the CoCoME

Modelling technique

**Modelling overview**

Selected issues

## ④ Analysis

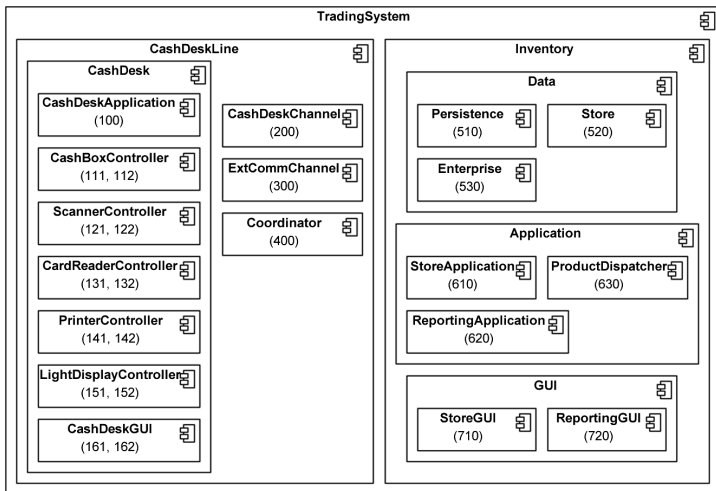
Temporal-logic properties

Use cases and test cases

## ⑤ Conclusion

# Modelling overview

- 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

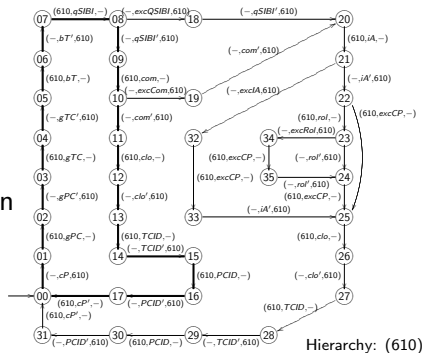
## ⑤ Conclusion

## Selected issues

1/2

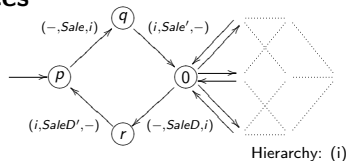
## Exception handling

- try, catch, finally blocks
- throw, delegate an exception



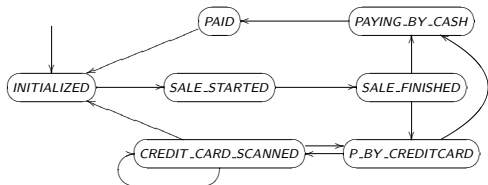
## Creation and destruction of instances

- 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

# Analysis

## Input for the analysis

- Model of the system as a [CI 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

# Temporal-logic properties

## Logic for expressing properties CI-LTL

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

## 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.

$\mathcal{G} (\mathcal{P}(610, \text{beginTransaction}, 511))$

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

- 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.

$\mathcal{G} (\mathcal{E}(610, \text{queryStockItemById}, -))$

$\Rightarrow \mathcal{F} \mathcal{E}(610, \text{queryStockItemById}, 521))$

- 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

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

- (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

**Thank you** for your attention

