Modelling the CoCoME with the Java/A Component Model

Rolf Hennicker, Alexander Knapp
Ludwig-Maximilians-Universität München

August 2007

The Java/A Team

- Ludwig-Maximilians-Universität München
  - UML modelling, qualitative analysis

  - Florian Hacklinger
  - Rolf Hennicker
  - Stephan Janisch
  - Alexander Knapp
  - Martin Wirsing

- University of Edinburgh
  - quantitative analysis

  - Allan Clark
  - Stephen Gilmore

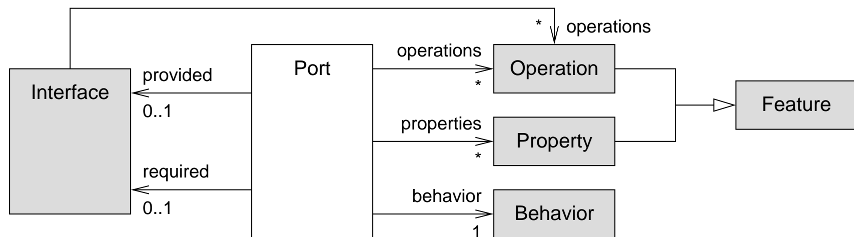
- Danmarks Tekniske Universitet, Lyngby
  - UML modelling, qualitative analysis

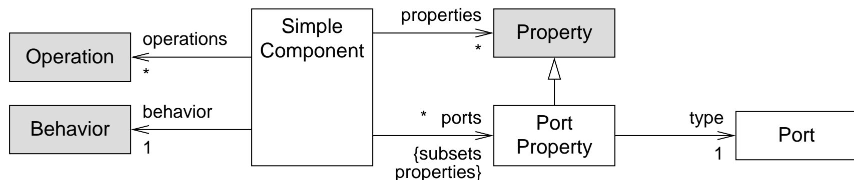
  - Hubert Baumeister

Overview

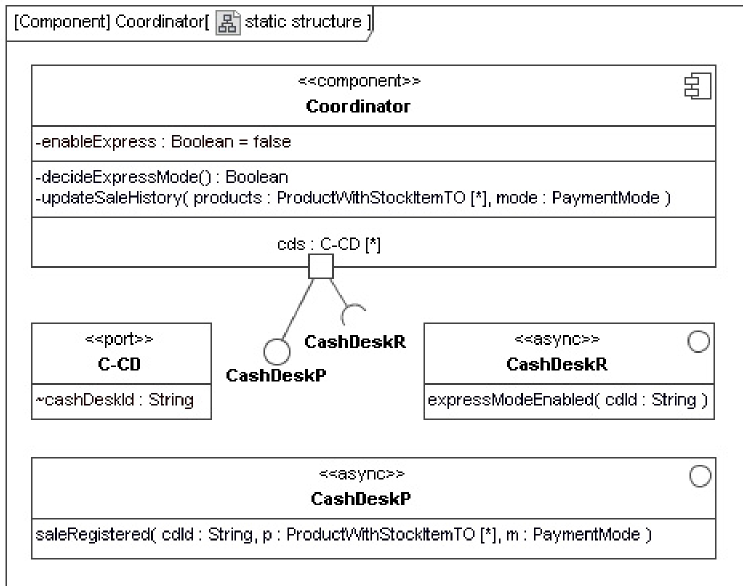
- Java/A component model
- Modelling the CoCoME
- Analysis of our model for the CoCoME
- Tools
- Open issues

The Java/A Component Model (1)

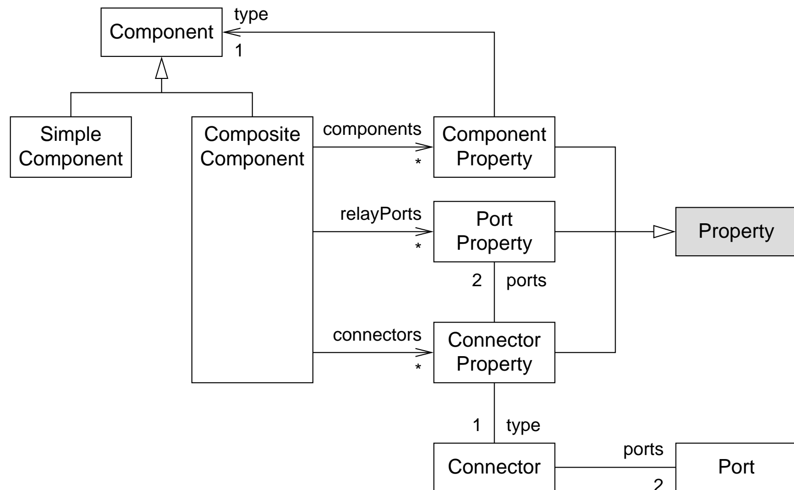




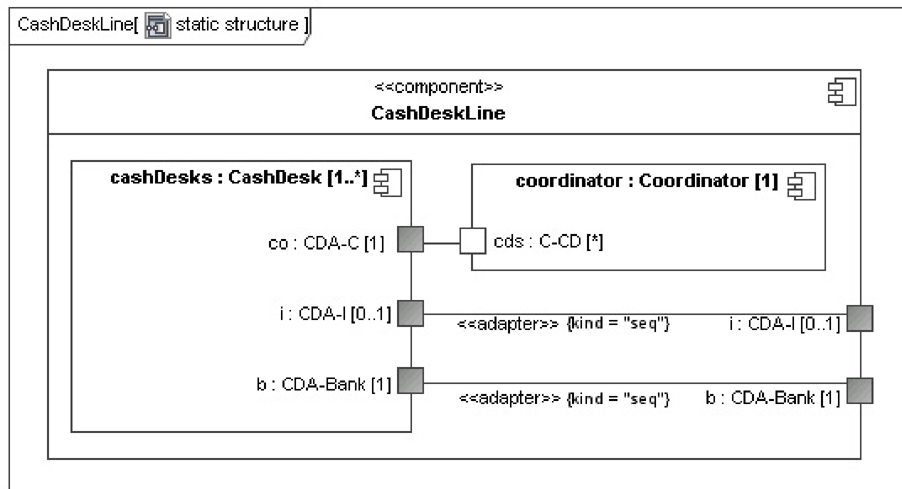
Simple Components: Example



The Java/A Component Model (2)



Composite Components: Example

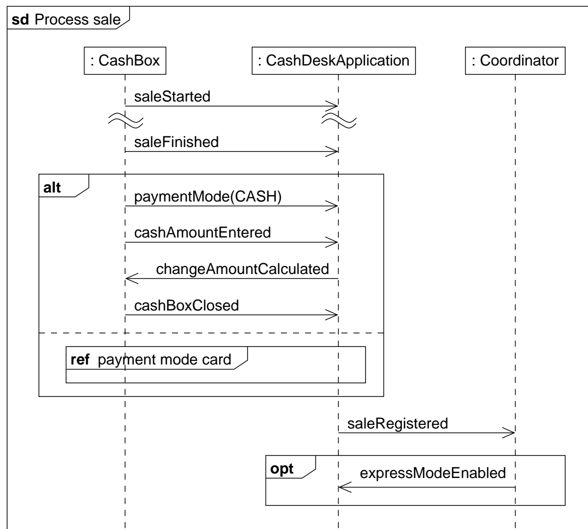


Modelling the CoCoME — Procedure (1)

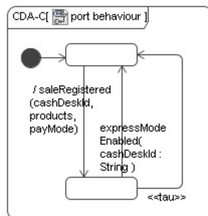
1. Primary scenarios from CoCoME description
   - static structure of simple components
   - port and component behaviour
   - analysis of behaviour (embedded system part)
   - integration of simple components into composite components
2. Alternative and exceptional scenarios from CoCoME description
3. Iteration

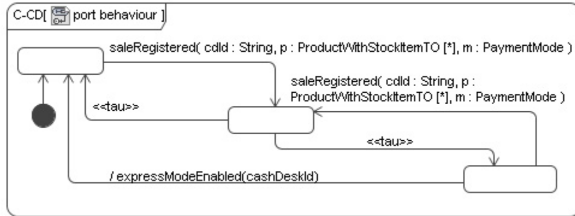
Modelling the CoCoME — Procedure (2)

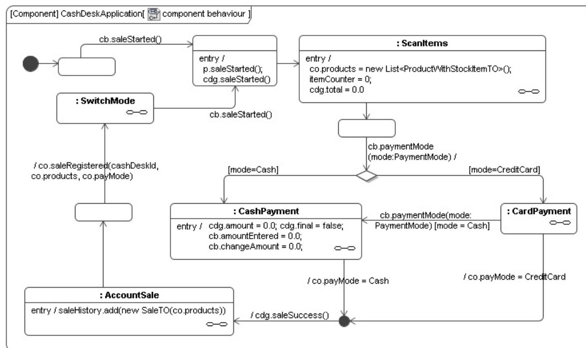
- Deriving state machines from interactions



Modelling the CoCoME — Procedure (3)



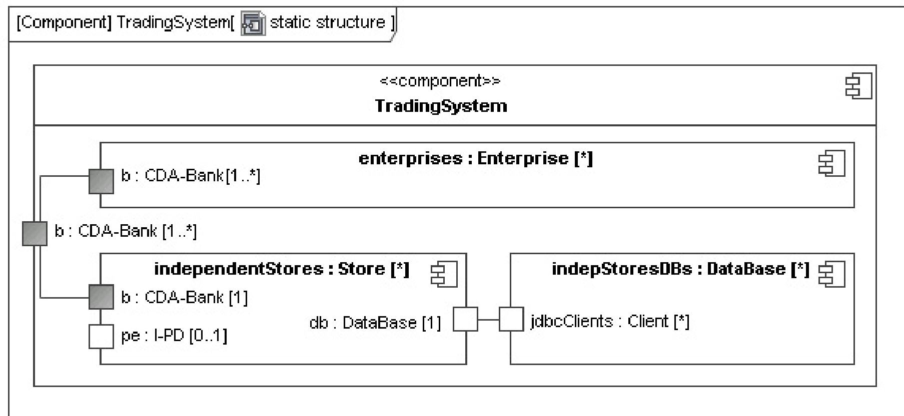




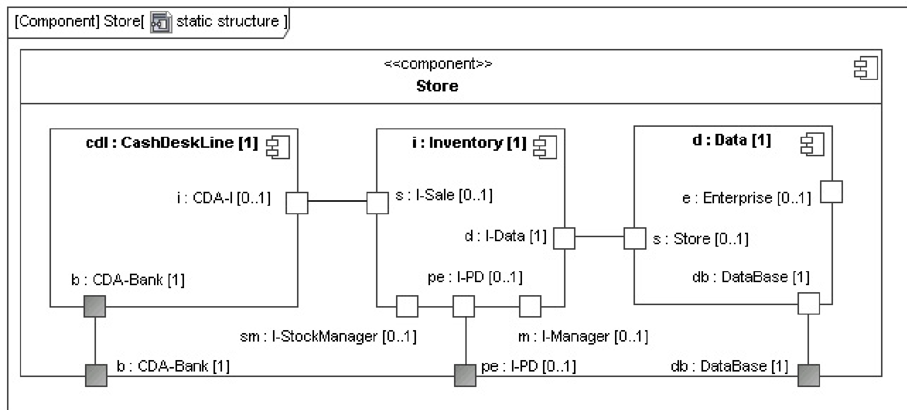
Modelling the CoCoME — Overview

CoCoME	Model
CashDeskLine	Store::CashDeskLine
CashDesk	CashDesk
CardReaderController	CardReader
CashBoxController	CashBox
CashDeskApplication	CashDeskApplication
CashDeskGUI	CashDeskGUI
LightDisplayController	LightDisplay
PrinterController	Printer
ScannerController	Scanner
Coordinator	Coordinator
EventBus	Not modelled
Inventory	Not explicitly; distinguished Enterprise/Store
Application	Ditto due to distinct Enterprise/Store
Reporting	Enterprise::Reporting
Store	Store::Inventory
Data	Data
Enterprise	Data instantiated in Enterprise
Store	Data instantiated in Store
Persistence	integrated in ports of Data
GUI	modelled as operator ports
Reporting	ports of Enterprise::Reporting
Store	ports of Store::Inventory
DataBase	DataBase

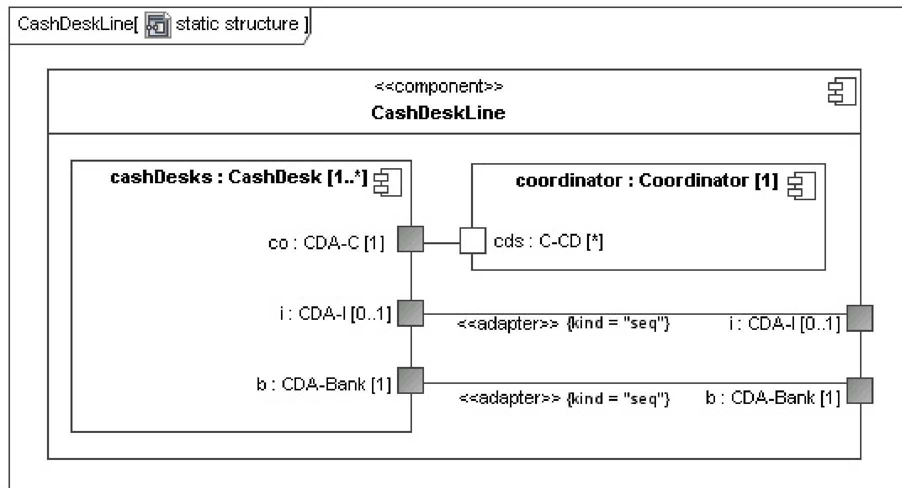
TradingSystem — Static Structure



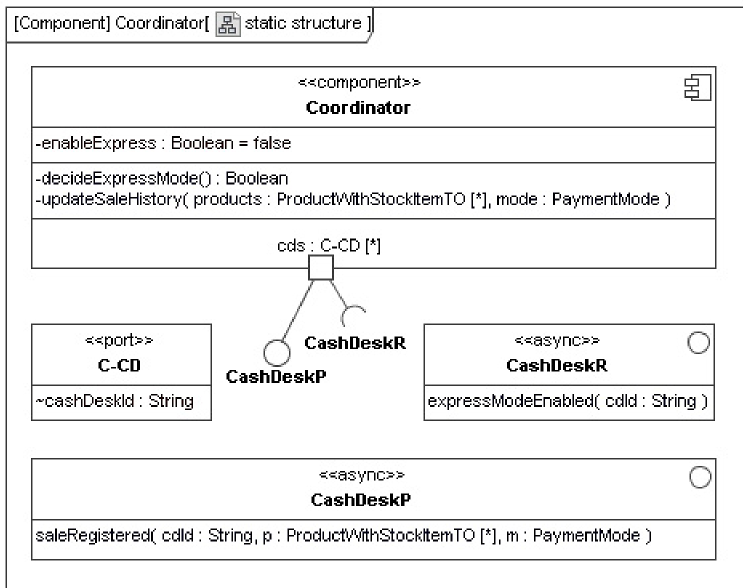
Store — Static Structure



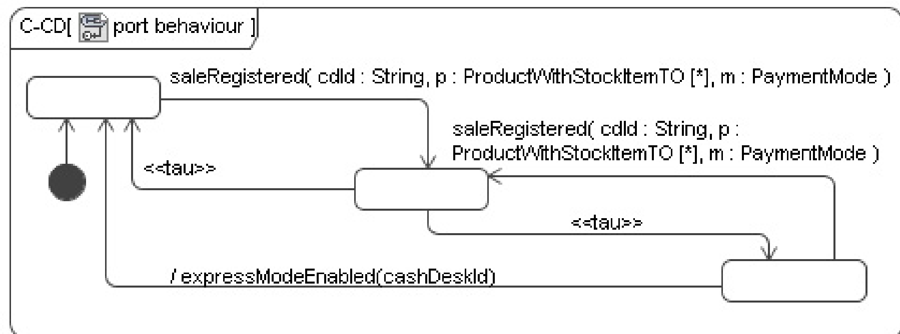
CashDeskLine — Static Structure



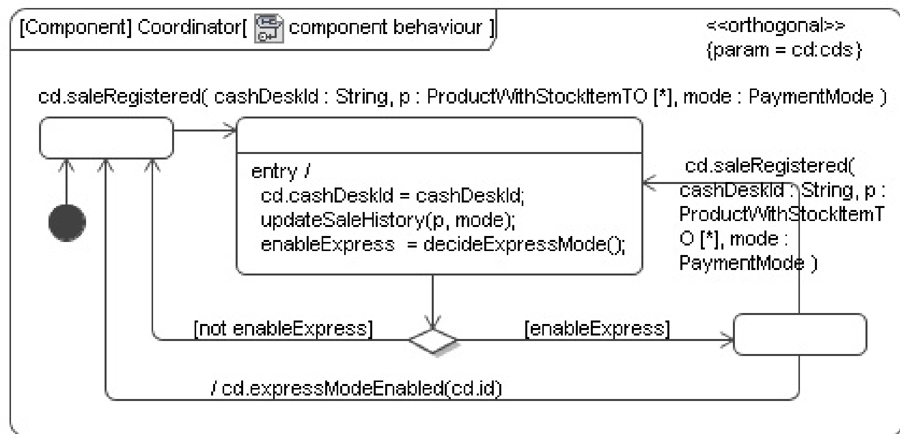
Coordinator — Static Structure



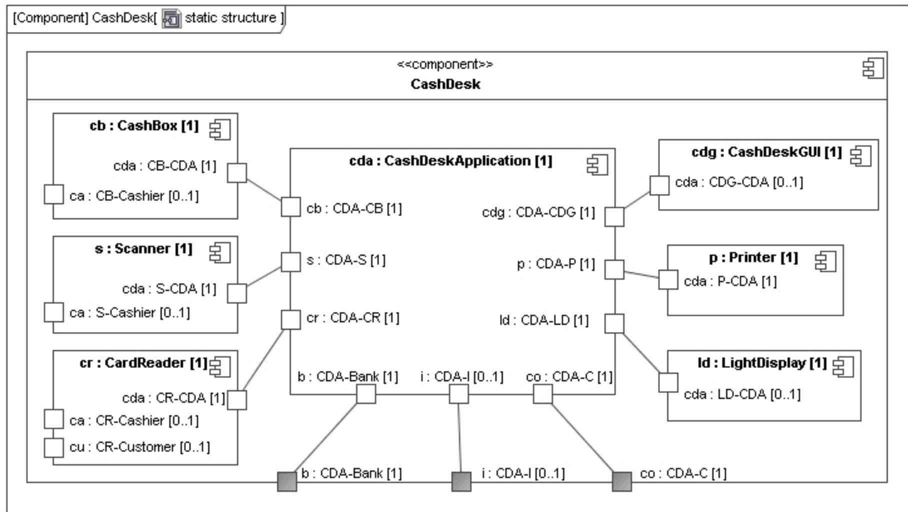
Coordinator — Port Behaviour



Coordinator — Component Behaviour



CashDesk — Static Structure



Functional Analysis

Interested in:
Behaviour of ports and components (simple and composite)

Basis:
Given *behaviour specifications* of ports and *simple* components

Properties to be checked:

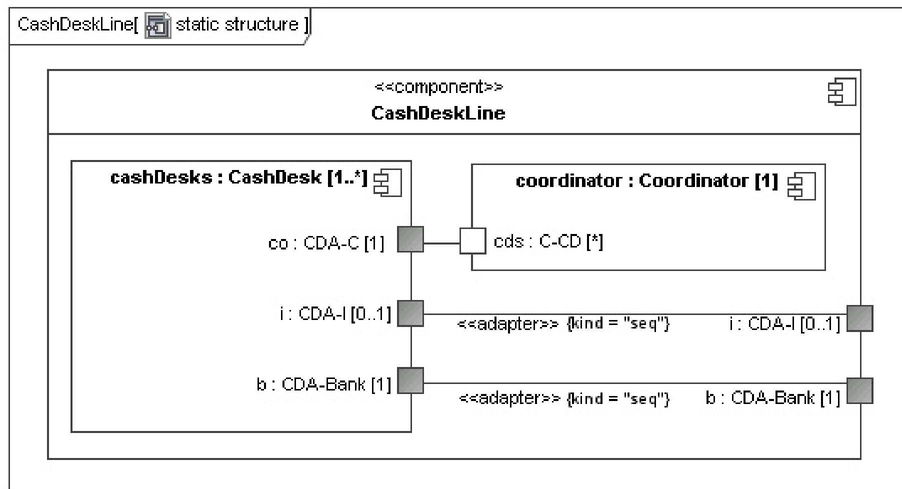
- *Deadlock-freeness* of port and component behaviours
- *Correctness* of components w.r.t. their ports

Focus of our analysis: Embedded system part

Analysis process:

- Starts with the analysis of simple components and their ports
- Derives results for composite components

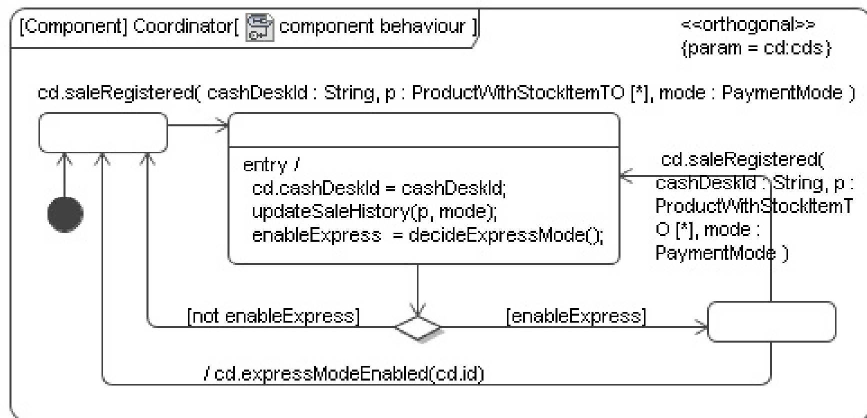
Composite Component CashDeskLine (revisited)



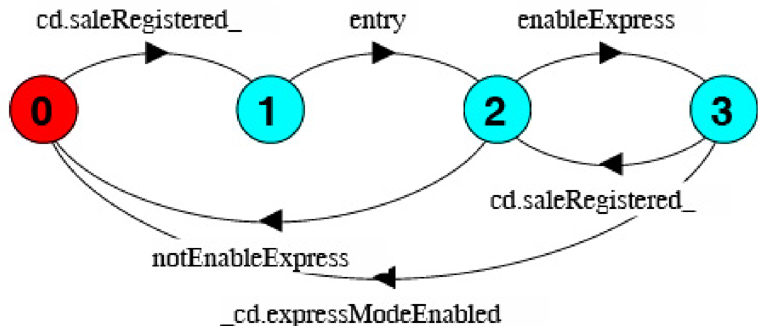
Behaviour Specifications and their Formalisation

State machines $\mapsto$ I/O-transition systems

with *input, output, internal* labels + τ-action



I/O-transition System for Component Coordinator



Notation: beh(Coordinator)

Behaviour of Port Coordinator–CashDesk

C-CD[port behaviour]

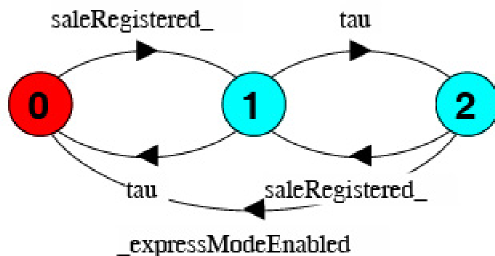
saleRegistered(cdId : String, p : ProductWithStockItemTO [*], m : PaymentMode)

saleRegistered(cdId : String, p : ProductWithStockItemTO [*], m : PaymentMode)

<<tau>>

<<tau>>

/ expressModeEnabled(cashDeskId)



Analysis of Simple Components

Steps:

- Check the deadlock-freeness of ports and simple components
- Check the compliance of component and port behaviours

Definition (*Component correctness*)

Observable behaviour of a component C *at port* $p : P$

$$obs_p(beh(C)) \quad \approx \quad beh(P)$$

Example: $obs_{\mathsf{cd}}(beh(\text{Coordinator})) \approx beh(\text{C-CD})$

Analysis of Composite Components

Assume given: Correct and deadlock-free subcomponents

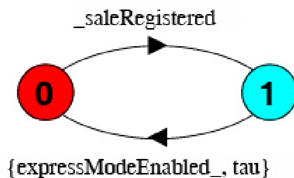
Analysis steps:

- Examine (pairwise) the interaction behaviour of connected ports
- Check the deadlock-freeness of the composite component
- Check the correctness of the composite component w.r.t. its ports

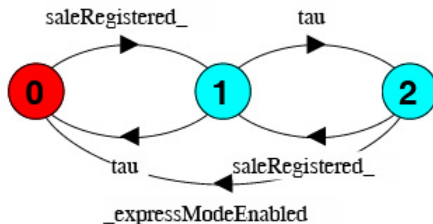
Interaction Behaviour of Connected Ports

Example:
Interaction behaviour of the Coordinator and CashDesk ports

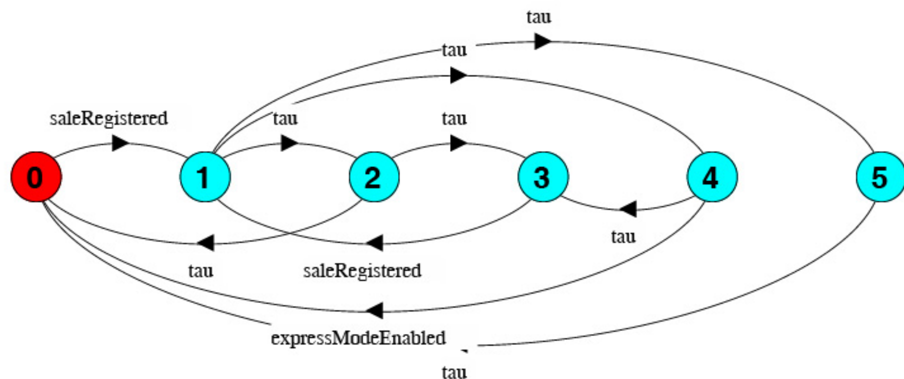
beh(CDA-C)



beh(C-CD)



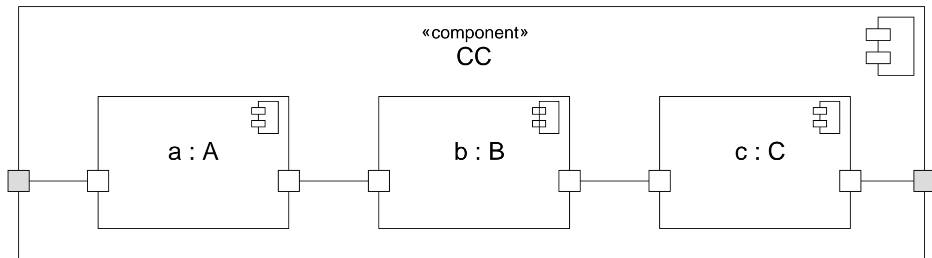
Port product $beh(\text{CDA-C}) \otimes beh(\text{C-CD})$



Definition (*Behavioural compatibility of ports*)

$beh(P) \otimes beh(Q)$ is deadlock-free.

Important Observation



Behavioural compatibility of the connected ports

$+$ deadlock-freeness of all subcomponents

$\not\Rightarrow$

deadlock-freeness of the composite component

Reflection of Port Behaviour

Definition (*Reflection of port behaviour*)

The interaction behaviour of P and Q *reflects* the behaviour of P, if

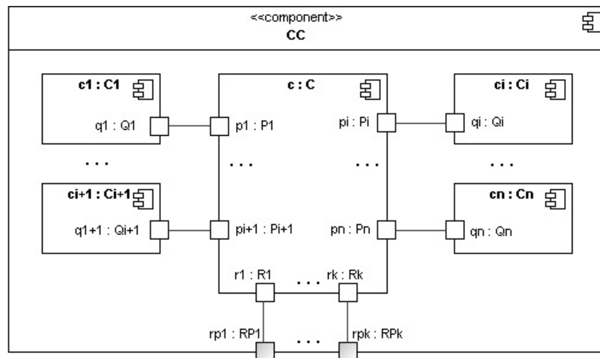
$$beh(P) \;\approx\; beh(P) \otimes beh(Q)$$

Example:

The interaction behaviour of the $CashDesk$ and the $Coordinator$ ports reflects the behaviour of the $CashDesk$ port

$$beh(\text{CDA-C}) \;\approx\; beh(\text{CDA-C}) \otimes beh(\text{C-CD})$$

Deadlock-freeness of Composite Components

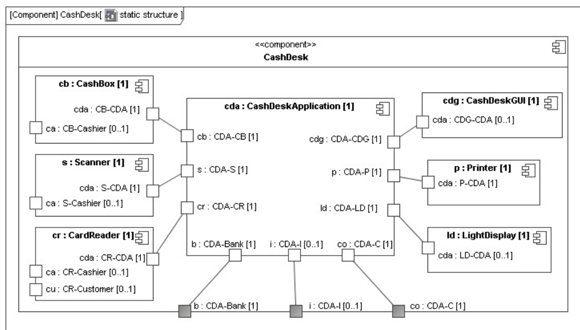


Assume:

- Deadlock-freeness and correctness of all subcomponents
- Behavioural compatibility of all connected ports
- Behaviour reflection for $n - 1$ connected ports

Then the composite component CC is deadlock-free.

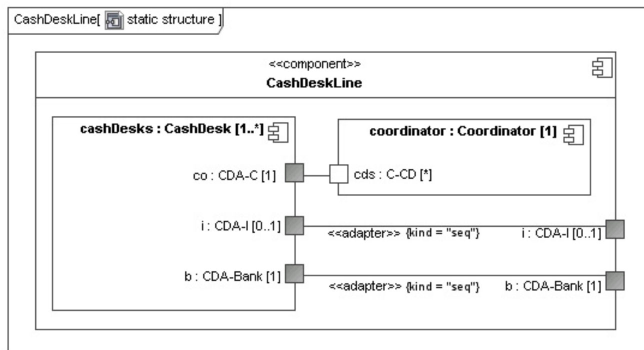
Deadlock-freeness of the CashDesk Component



- All subcomponents are deadlock-free and correct
- All connected ports are behaviourally compatible
- Only the connection between the CashDeskApplication and the CashBox is not behaviour reflecting

Hence the CashDesk component is deadlock free.

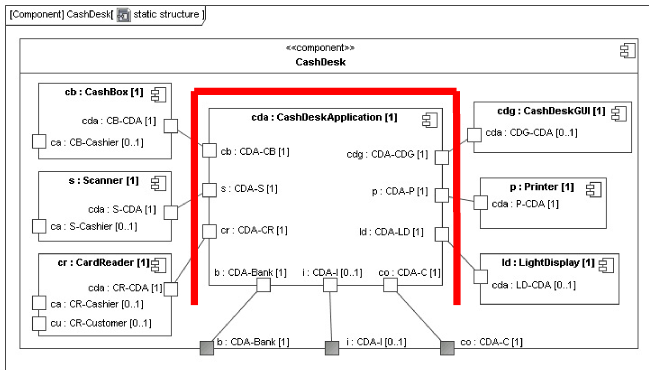
Deadlock-freeness of the CashDeskLine



- The subcomponents Coordinator and CashDesk are deadlock-free and correct
- The connected ports are behaviourally compatible (and even behaviour reflecting)

Hence the CashDeskLine component is deadlock free.

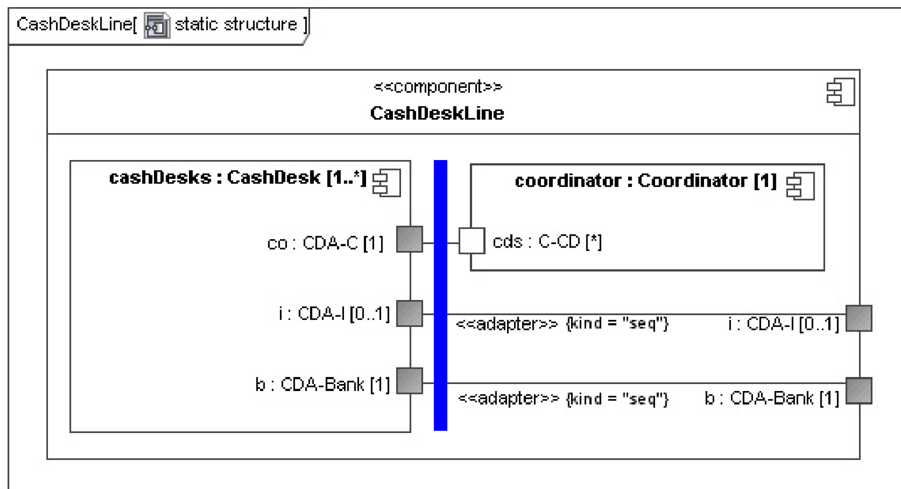
Implementation Model with Event Bus (1)



Claim

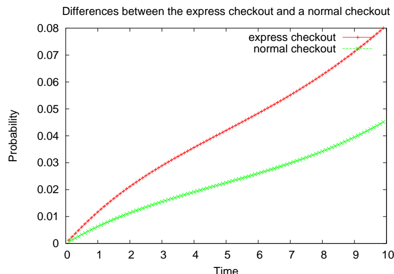
- If the event bus works in FIFO manner, communication order is preserved.
- Deadlock-freeness is preserved.

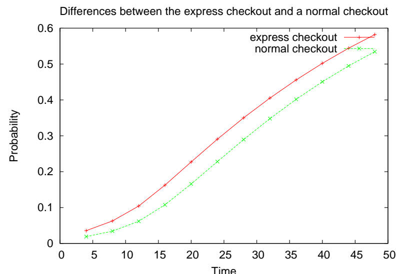
Implementation Model with Event Bus (2)



Non-Functional Analysis

- Assessment of Service-Level Agreements
  - currently to be done manually (PEPA)
  - only exemplified for express checkout
- Assessment of advantage of using express checkout
  - customers with eight items or fewer may also use normal checkout





Tools

- MagicDraw
  - UML modelling
- Labelled Transition System Analyser (LTSA)
  - based on process algebra FSP
  - used for analysis of deadlock and observational equivalence
- Hugo/RT
  - UML model translator for model checking (SPIN, UPPAAL), theorem proving (KIV), and code generation (Java, SystemC)
  - used for code generation for Java/A
- Imperial PEPA Compiler (IPC)
  - based on Performance Evaluation Process Algebra (PEPA)
  - used for quantitative analysis with Continuous-Time Markov Chains

Java/A: Architectural Programming

- Architectural programming language
  - integration of architectural notions into Java
  - avoiding architectural erosion in implementation

```
composite component SimplifiedStore {
  assembly {
    components {
      Inventory, CashDesk
    }
    connectors {
      Inventory.Sale, CashDesk.CDAI;
    }
  }
  constructor SimplifiedStore() {
    initial configuration {
      active component Inventory inv = new Inventory();
      active component CashDesk cd = new CashDesk();
      connector Connector con = new Connector();
      con.connect(inv.Sale, cd.CDAI);
    }
  }
}
```

Java/A: Components and Ports

```
simple component CashDeskApplication {
  int itemCounter = 0; ...
  port CDACB {
    provided { async saleStarted();
               async productBarCodeEntered(int barcode);
               async saleFinished();
               async paymentModeCash(); ... }
    required { async changeAmountCalculated(double amount);
               async saleSuccess(); }
    protocol <! behaviour {
      states { initial init;
               simple a; simple b; simple e; ... simple h; }
      transitions { init -> a;
                    a -> b { trigger saleStarted; }
                    b -> b { trigger productBarCodeEntered; }
                    ...
                    e -> h { effect out.saleSuccess(); }
                    h -> b { trigger saleStarted; }
                  } } !>
  }
  ...
```

Java/A: Interface Implementation

```
  void saleStarted() implements CDACB.saleStarted() {
    Event event = Event.signal("send saleStarted",
                               new Object[]{});
    this.eventQueue.insert(event);
  }
  ...
  void processSaleStarted() {
    try {
      CDAP.saleStarted();
      CDACDG.saleStarted();
    }
    catch (ConnectionException e) {
      e.printStackTrace();
    }
  }
  ...
}
```

Open Issues

- Extension of functional analysis
  - integration of pre- and post-conditions for synchronous operations
  - n-ary connectors
- Integration of non-functional analysis
  - annotation of state machines and sequence diagrams by performance properties
- Runtime reconfiguration
  - e.g., opening and closing cash desks
- Deployment view