CoCoME in Fractal

Lubomír Bulej, Tomáš Bureš, Martin Děcký, Pavel Ježek, Pavel Parízek,
František Plášil, Tomáš Poch, Nicolas Rivierre, Ondřej Šerý, Petr Tůma

DISTRIBUTED SYSTEMS RESEARCH GROUP
FACULTY OF MATHEMATICS AND PHYSICS
CHARLES UNIVERSITY, CZECH REPUBLIC
http://dsrg.mff.cuni.cz

FRANCE TELECOM R&D
ISSY LES MOULINEAUX, FRANCE
http://fractal.objectweb.org

Fractal Team Members

- Charles University DSRG
  - Software components
    - Architecture and component models (SOFA)
    - Formal specification of behavior
  - Performance evaluation
    - Regression benchmarking
    - Performance modeling

- France Telecom R&D
  - Software components
    - Architecture and component models (Fractal)

Fractal Component Model

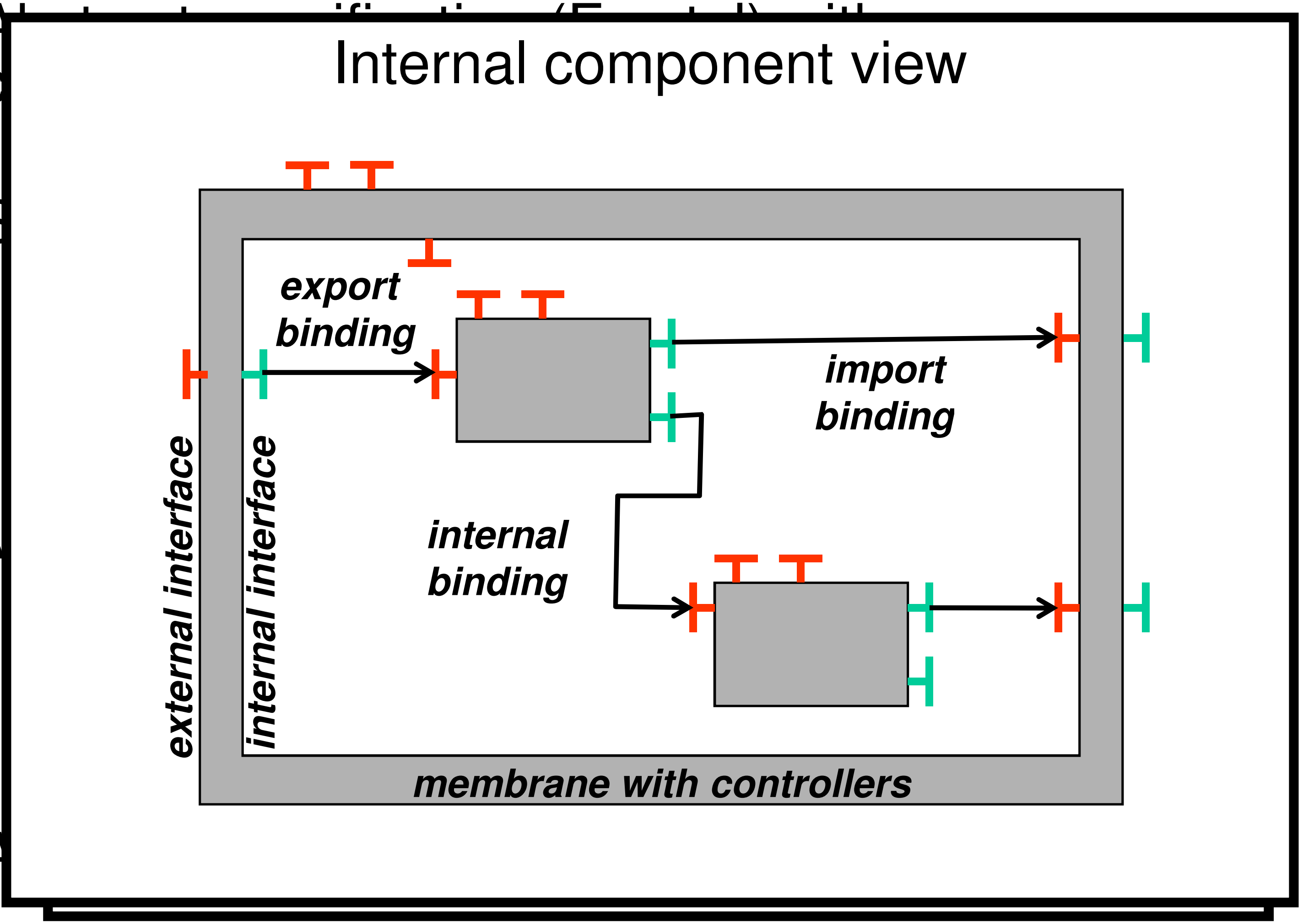
- Project hosted by OW2 consortium
- Lead development by INRIA, France Telecom R&D

- Complex applications ranging from embedded software to application servers and information systems

- Hierarchically composed components
- Shared components for resources
- Separation of concerns
  - Controller infrastructure
  - Runtime introspection
- Dynamic configuration and reconfiguration
- Behavior specification via Behavior Protocols
  - Composition correctness
  - Implementation compliance

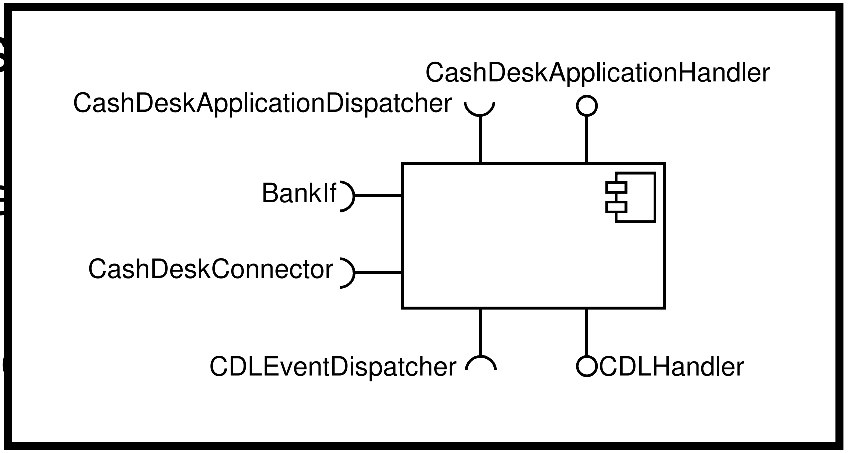
Static Architecture in Fractal

- A
  c
- S
- N
- A



Behavior Protocols in Fractal

- Process algebra expression des
  - Infinite set of finite event traces
  - Events are invocations on interface



- Fragment from CoCoME sale lo

```
# SALE_STARTED
(?CashDeskApplicationHandler.onProductBarcodeScanned
  {
    !CashDeskConnector.getProductWithStockItem ;
    !CashDeskApplicationDispatcher.sendBarcodeNotValid +
    !CashDeskApplicationDispatcher.sendRunningTotalChanged
  }
) * ;
?CashDeskApplicationHandler.onSaleFinished;
# SALE_FINISHED
```

Behavior Protocols Syntax

- Events
  - Emitting a method call request: **!interface.method↑**
  - Accepting a method call request: **?interface.method↑**
  - Emitting a method call response: **!interface.method↓**
  - Accepting a method call response: **?interface.method↓**

- Operators
  - Sequence **;**
  - Alternative **+**
  - Repetition *****
  - And-parallel interleaving |
  - Or-parallel interleaving | |
  - **Consent** ∇

    parallel composition (interleaving + internal events τ)

    can indicate communication errors

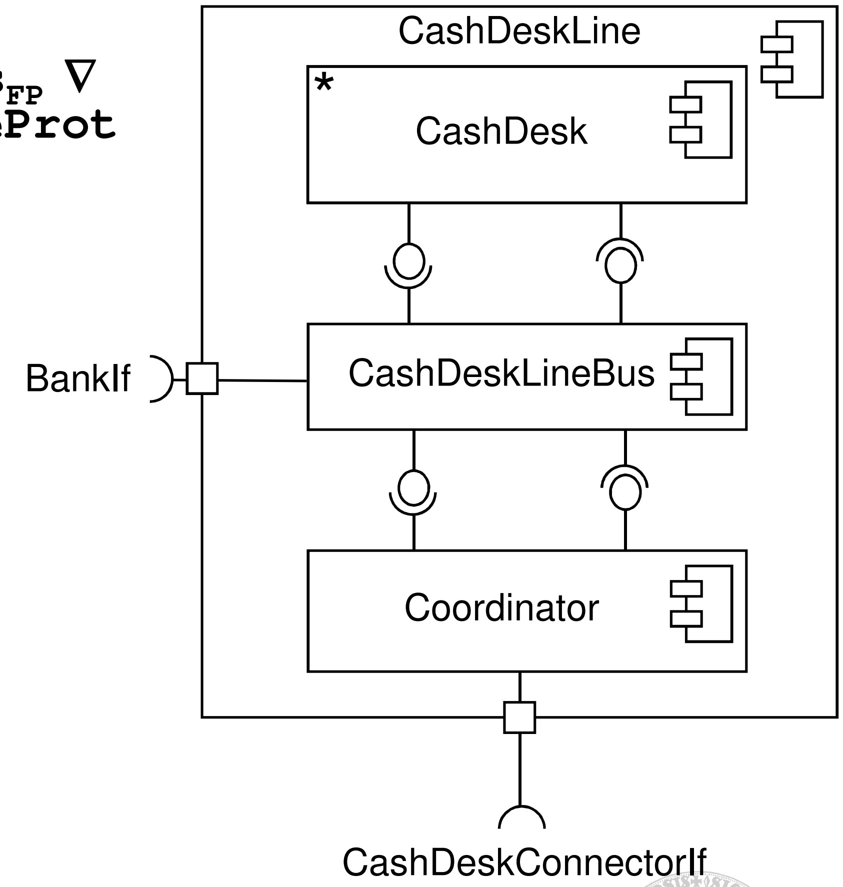
    no activity (deadlock)

    bad activity (emitted call cannot be accepted)

- Syntactic sugar for method internals
  - **?i.m** = **?i.m↑ ; !i.m↓**
  - **?i.m {***prot***}** = **?i.m↑ ;** *prot* **; !i.m↓**

Behavior Compliance Checking

- **Horizontal compliance**
  - Do the components at the same level cooperate correctly ?
  - $\mathbf{CashDesk}_{FP} \nabla \mathbf{CashDeskLineBus}_{FP} \nabla \mathbf{Coordinator}_{FP} = \mathbf{ArchitectureProt}$
- **Vertical compliance**
  - Does the composite component do what its interface claims ?
  - $\mathbf{ArchitectureProt} \nabla \mathbf{CashDeskLine}_{FP}{}^{-1}$
  - Both checked by Behavior Protocol Checker (BPC)
- **Implementation compliance**
  - Does the implementation do what its interface claims ?
  - Checked by a combination of Java Path Finder (JPF) and Behavior Protocol Checker (BPC)



Implementation Compliance with JPF and BPC

- JPF traverses the state space of the component implementation
  - Notification about method calls sent to BPC
  - Notification about backtracking sent to BPC
- BPC follows JPF
  - JPF method calls are BPC protocol state transitions
  - JPF backtracking causes BPC backtracking as well
- Missing environment problem
  - JPF only checks a complete program
  - We generate an artificial environment
    - All possible calls as prescribed by the protocol
    - Composition of component + environment checked

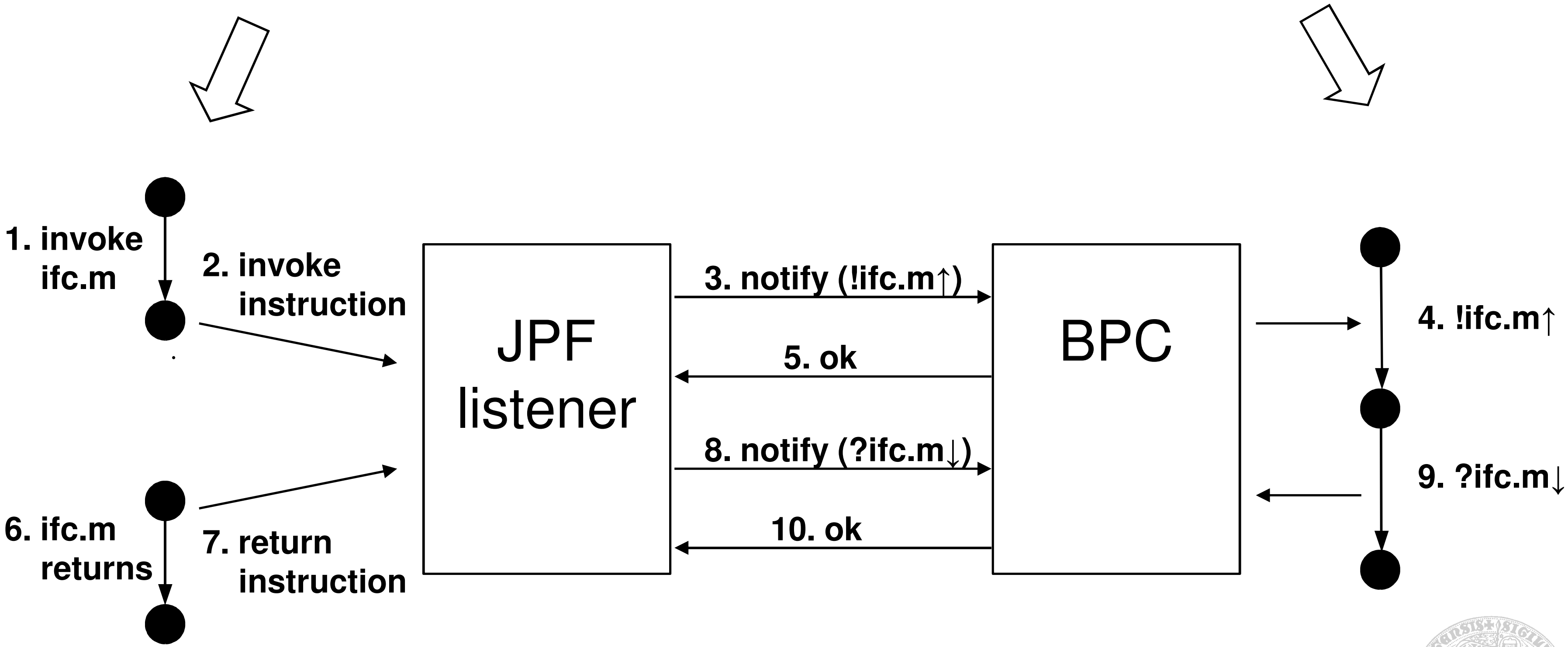
Communication Between JPF and BPC

JPF state space

Java code of
component + environment

BPC state space

protocol of component



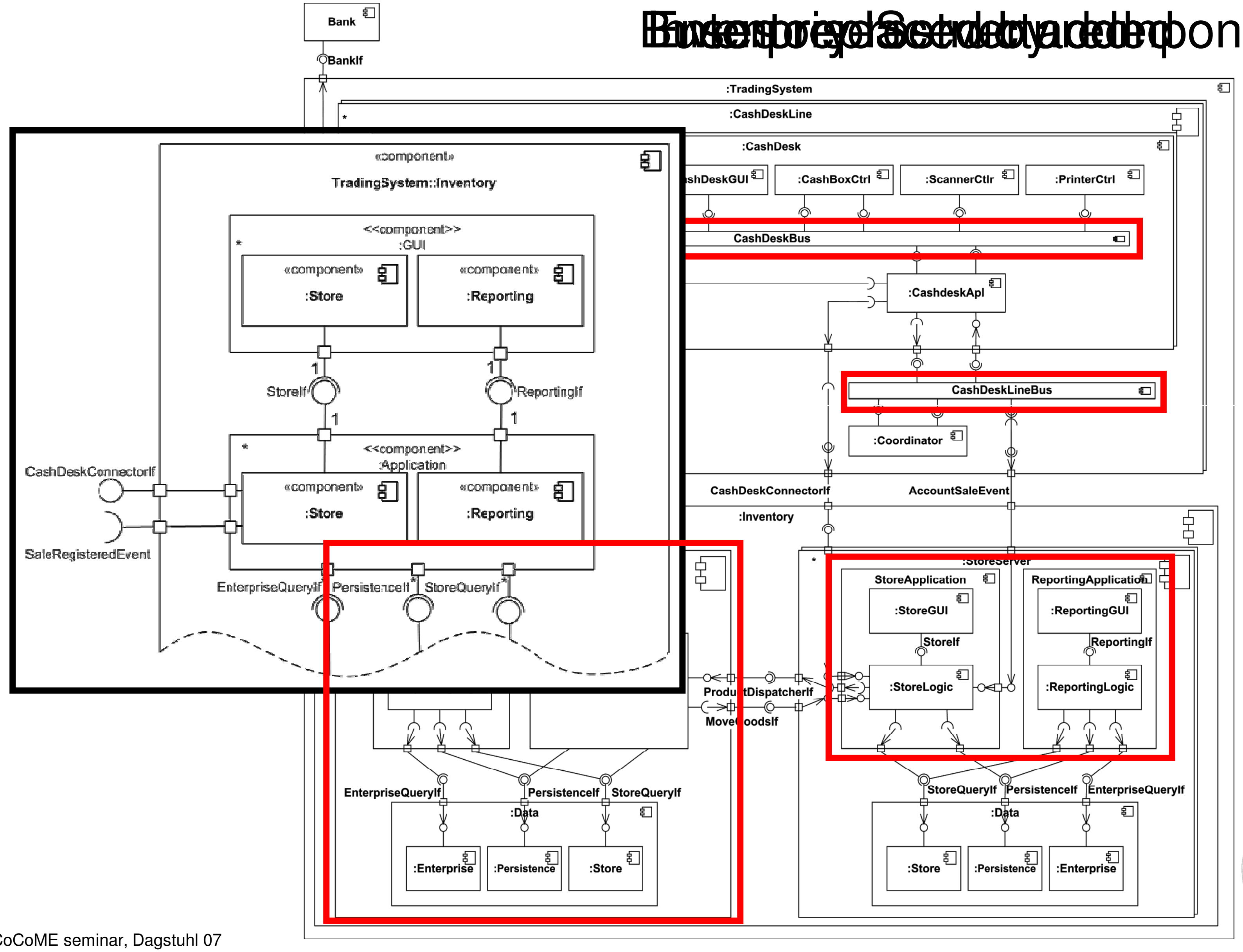
Modeling CoCoME in Fractal

- Created
  - Architecture captured in Fractal ADL
  - Behavior described in Behavior Protocols
  - Reference implementation converted using the Julia implementation of Fractal
- Benefits
  - Compliance of component behavior specification checked
  - Correspondence between component code and its behavior specification checked
  - Extra functional properties monitored transparently

Static Architecture View in Fractal ADL

- Mostly straightforward modeling
- Original architecture modified to
  - Correspond to Fractal abstractions
    - Buses replaced by components
  - Improve inventory structure
    - Restructured to remove redundant layer
  - Support UC-8
    - Explicit component for Enterprise Server

Fractal Architecture



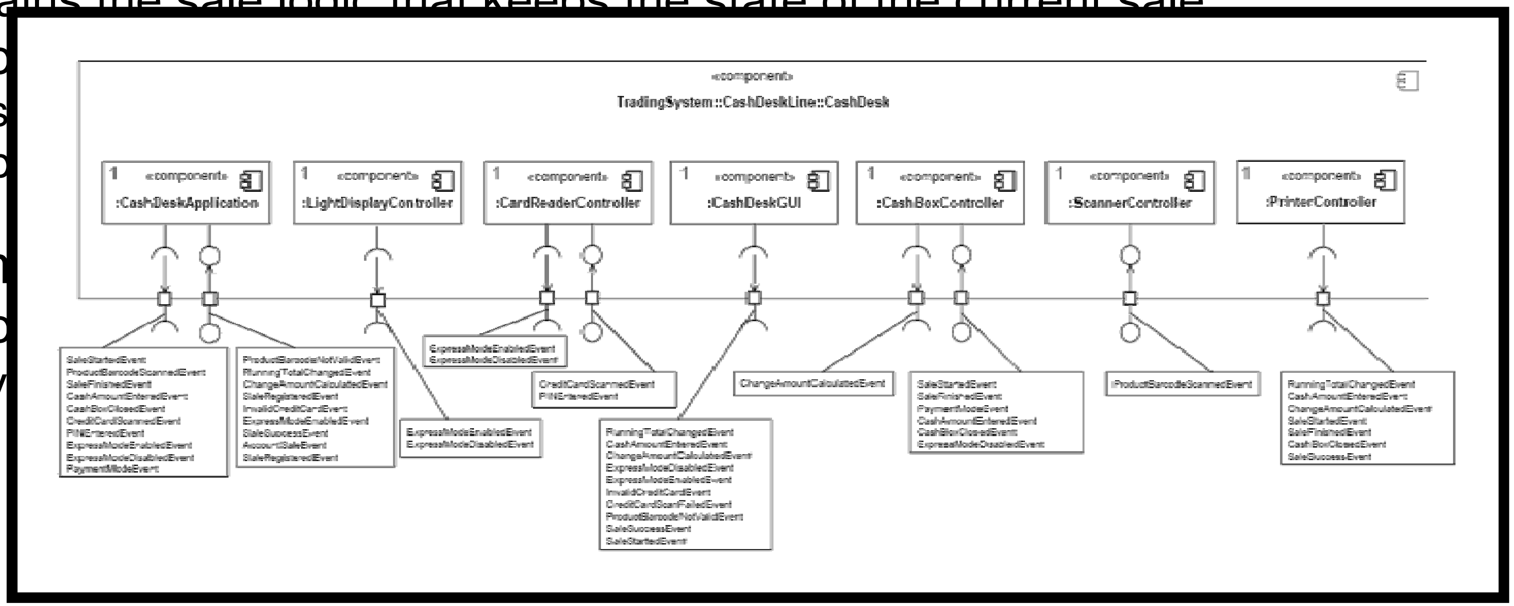
Approaches to Crafting Behavior Protocols

- BP integrates information from
  - multiple UML Sequence Diagrams, Use Case textual descriptions
  - reference implementation
  - additional design decisions

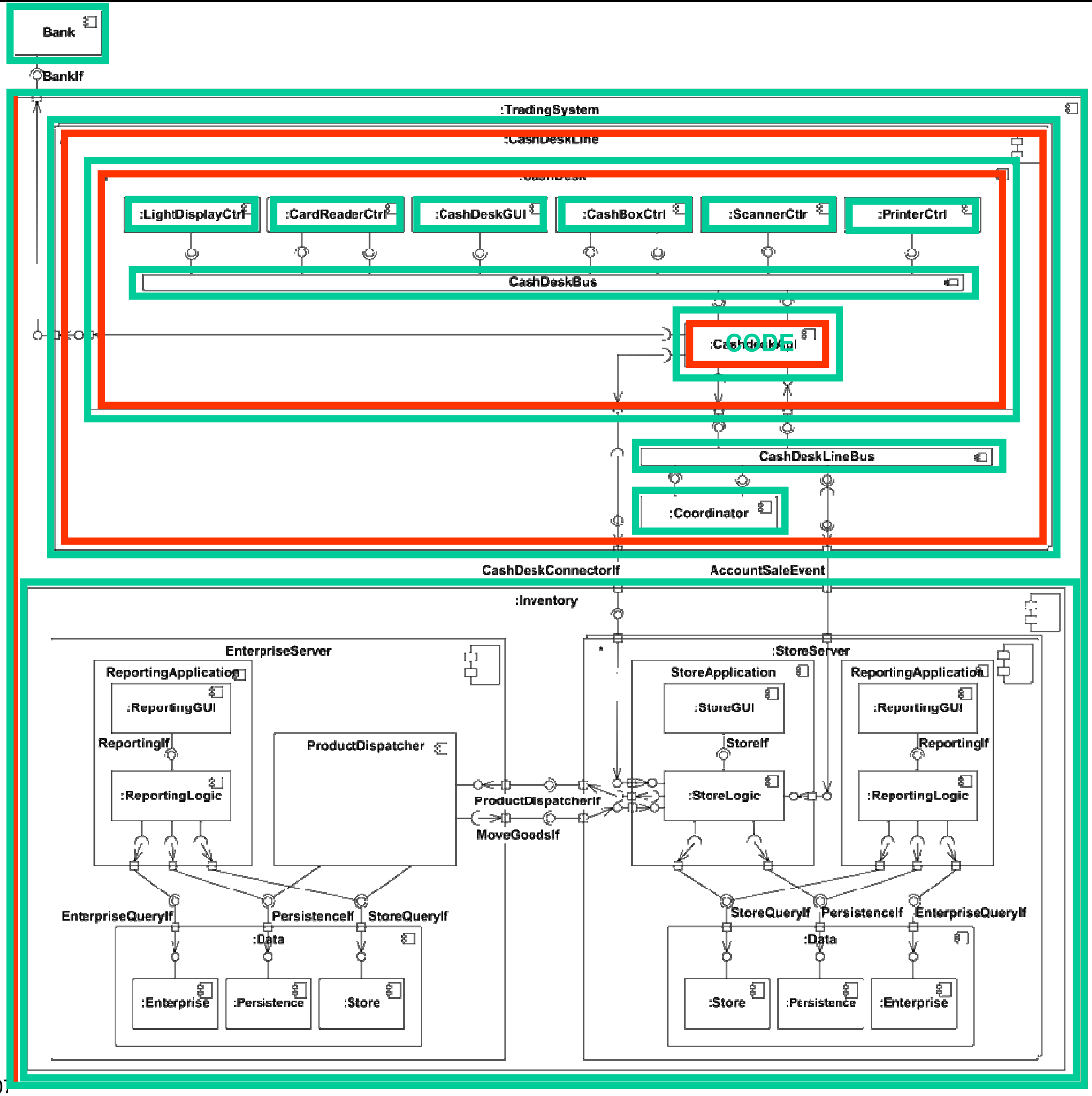
- Inventory components, CashDesk hardware
  - straightforward functionality, protocol derived from UML diagrams

- CashDeskApplication component
  - contains the sale logic that keeps the state of the current sale
  - proto
    - s
    - p

- Bus con
  - proto
  - deriv



Checking Compliance of Components



Checking of Primitive Components

- CashDeskApplication
  - Selected as it has the most complex behavior
  - We did not check other primitive components
- JPF requires complete program
  - Java environment created in two steps
    - Generated from the frame protocol
    - Manually modified to include arguments
- Discovered inconsistency of reference implementation wrt UC-1
  - Implementation trapped in a loop when the customer pays with invalid credit card
  - Discovered in **2** seconds !
  - Adjusted behavior checked in 14 seconds to challenge method feasibility

Checking Compliance of Components

- Component hierarchy
  - Splits the checking of the application into feasible subtasks
  - Each composite component checked independently

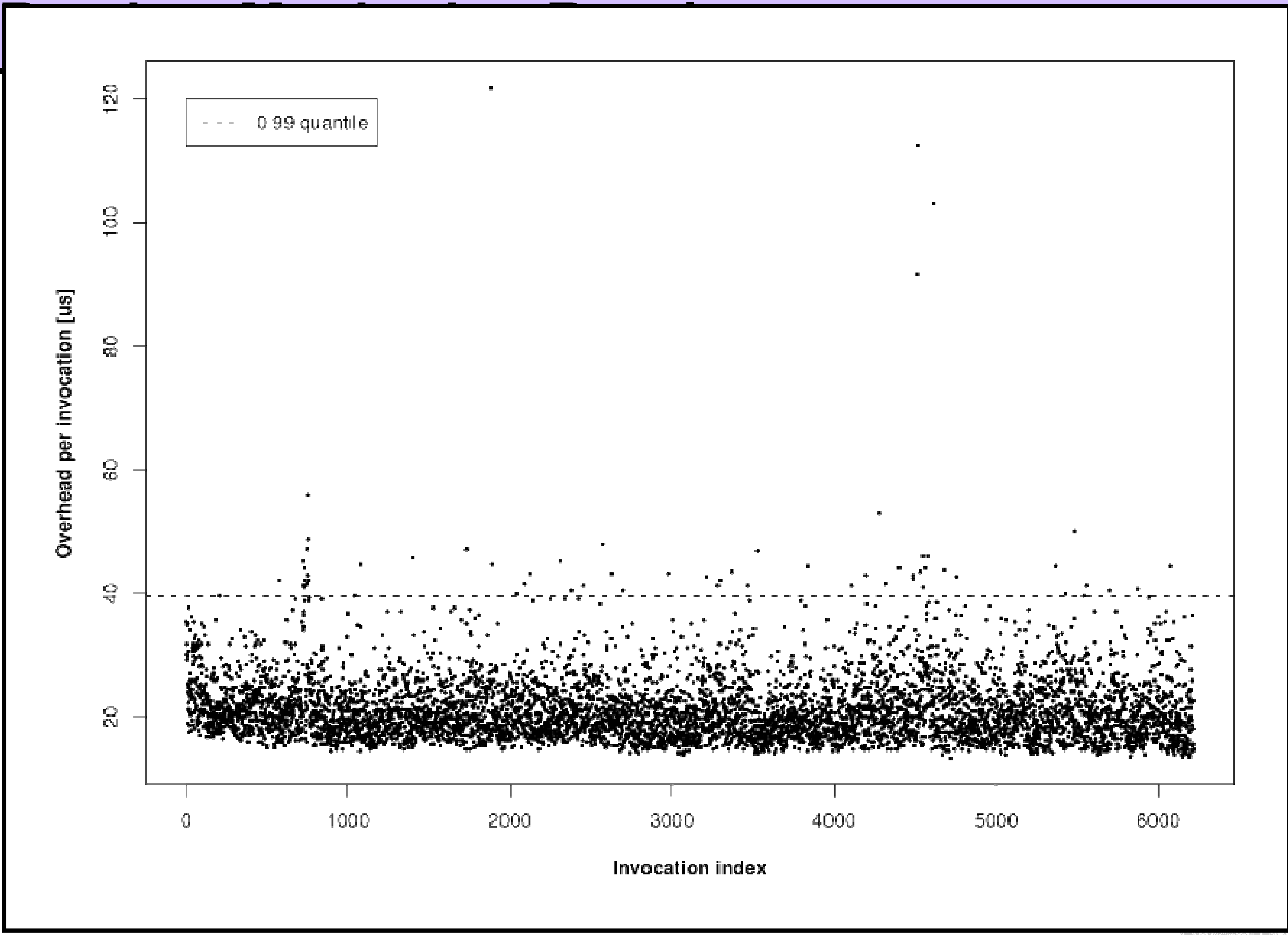
- Compliance of the whole Trading System was successfully checked

(Times for 2 x Core 2 Duo 2.3GHz, 4GB RAM)

Component	**Time [s]**	**States**
CashDesk	9.2	483,797
CashDeskLine	24.5	1,562
StoreApplication	6.9	63,900
Data	45.9	124,416
ReportingApplication	0.2	17
StoreServer	40.1	297,024
EnterpriseServer	39.5	512
Inventory	0.2	121
TradingSystem	18.0	51,558
Total time	184.5	1,022,907

Runtime Monitoring Overview

- Demonstrates capabilities of the component framework
- We focus on observation of extra-functional properties
  - Does the implementation work within the required limits ?
  - Do the external services meet the service level agreements ?
- Declarative configuration of monitoring infrastructure
  - Fractal configuration file describes controllers
  - Interceptor code generated transparently at runtime
  - Infrastructure accessible via standardized interfaces (JMX)
- Distinguishing features
  - Very low overhead
  - No modification of the application
  - Can observe any property at component level

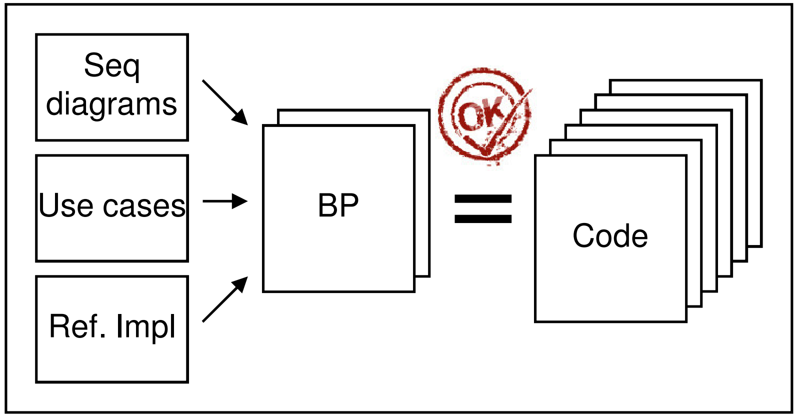

0.99 quantile
Overhead per invocation [us]
Invocation index
120
100
80
60
40
20
0
1000
2000
3000
4000
5000
6000

Conclusion

- Static view
  - The (slightly modified) architecture captured in Fractal
  - Buses replaced by components
    - No problems with synchronous communication
    - Asynchronous delivery difficult to model in BPs
    - Approximation using explicit buffers but awkward results
  - Intention to preserve the original architecture as much as possible did not pay off
    - We should have made more changes
    - Developers would do them during iterations anyway
- Runtime monitoring
  - Fully transparent monitoring
  - Can be used to check or enforce service level agreements

Conclusion

- BP versus UML
  - BP integrates
    - Number of UML Sequence Diagrams
    - Use Case textual descriptions
    - Reference implementation
  - BP captures
    - all traces corresponding to a particular start call in a sequence diagram
    - component hierarchy



- Static verification
  - feasible steps
    - protocol compliance
    - verification of code against frame protocols

Thank You

http://dsrg.mff.cuni.cz/cocome