CoCoME in SOFA 2.0

Tomáš Bureš, Martin Děcký, Petr Hnětynka, Jan Kofroň, Pavel Parízek,
František Plášil, Tomáš Poch, Ondřej Šerý, Petr Tůma

DISTRIBUTED SYSTEMS RESEARCH GROUP
http://dsrg.mff.cuni.cz

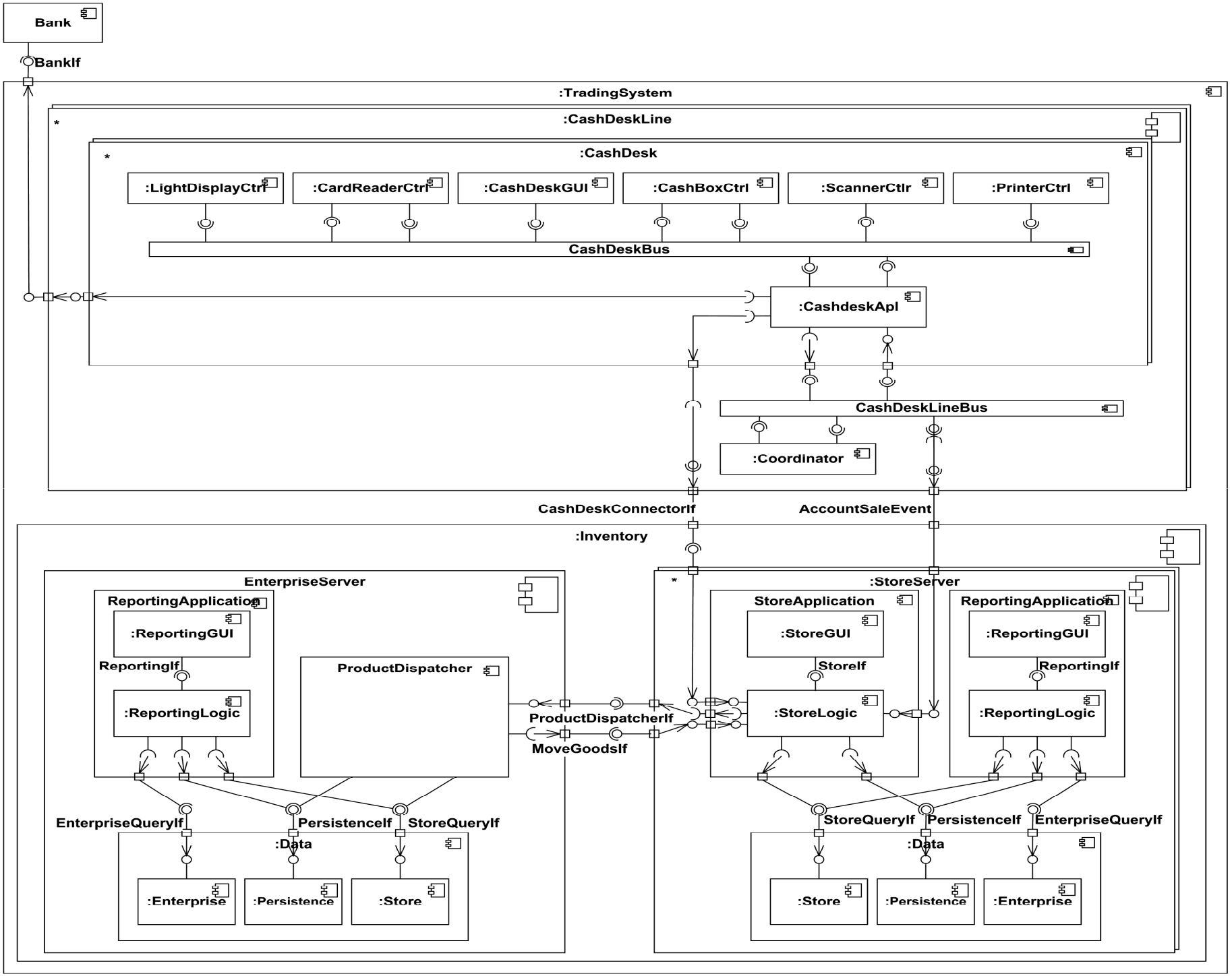
CHARLES UNIVERSITY PRAGUE
Faculty of Mathematics and Physics

Distributed Systems Research Group

- 3 key research topics
  - Components
    - SOFA, SOFA 2.0
  - Performance evaluation
    - Regression benchmarking
    - Performance modeling
  - Formal methods
    - Behavior modeling
      - Behavior Protocols (BP, EBP)
    - Code model checking
      - against BP

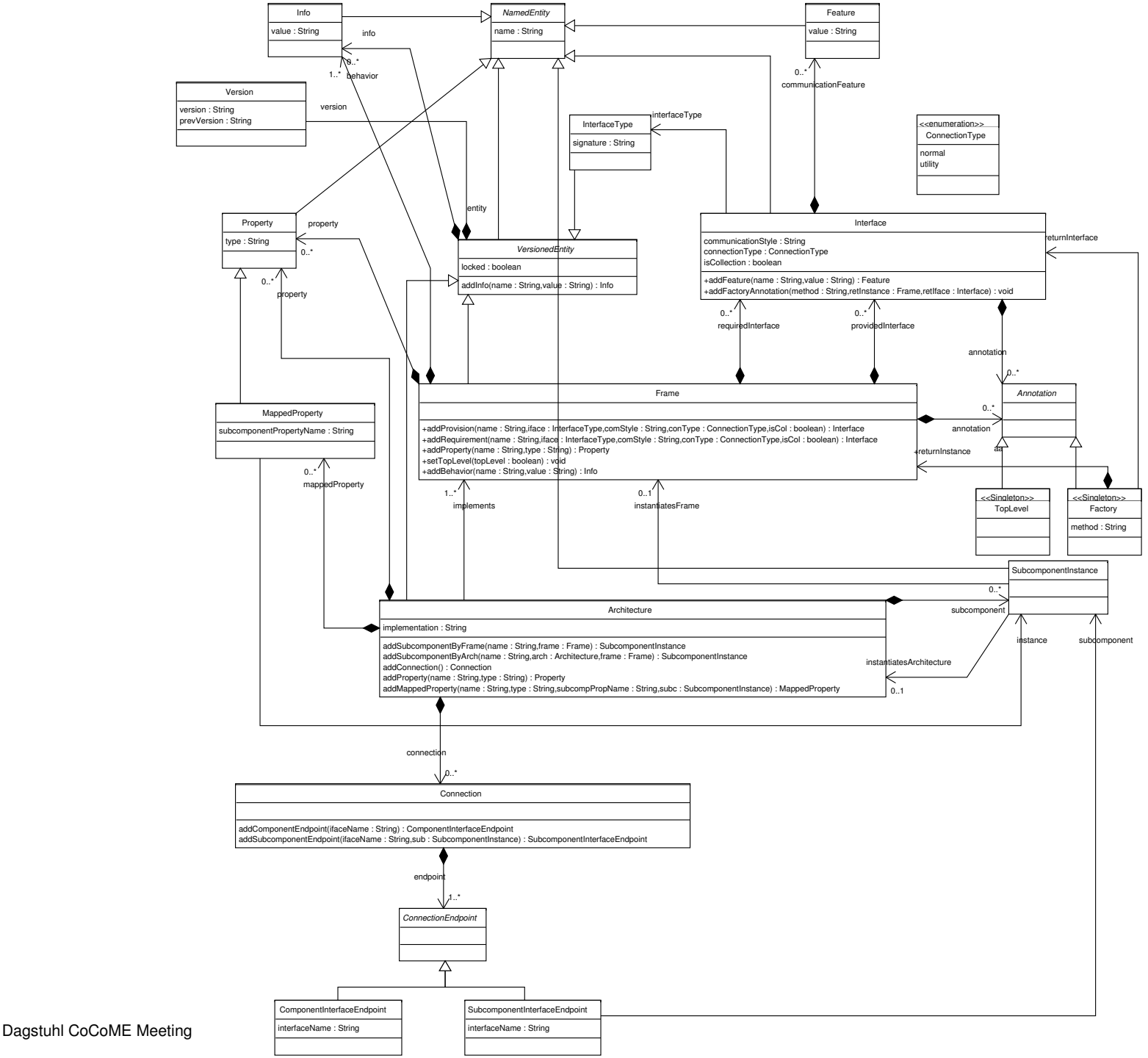
SOFA 2.0

- Hierarchical component model
  - Primitive and composite components
    - ADL includes
      - behavior spec (EBP)
      - utility interfaces (accessing services)
  - Connectors
    - communication styles (→ distribution)
  - Run time support
    - SOFANode
  - SOFA meta-model (using MOF)
    - Automated generation of
      - a repository
      - editing tools

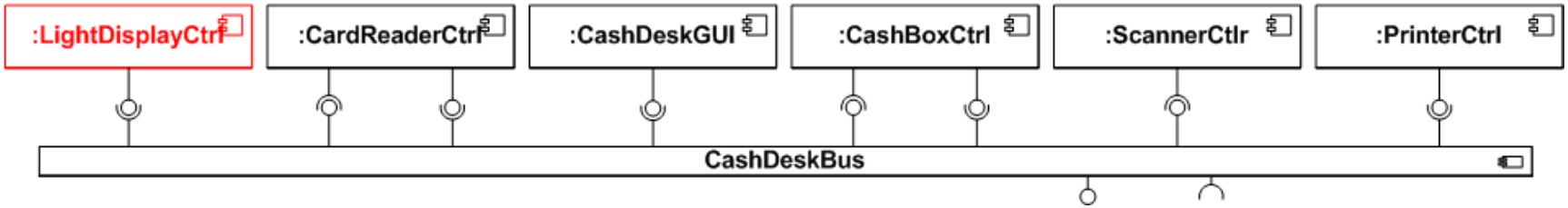

Bank
BankIf
:TradingSystem
:CashDeskLine
*
:CashDesk
*
:LightDisplayCtrl
:CardReaderCtrl
:CashDeskGUI
:CashBoxCtrl
:ScannerCtlr
:PrinterCtrl
CashDeskBus
:CashdeskApl
CashDeskLineBus
:Coordinator
CashDeskConnectorIf
AccountSaleEvent
:Inventory
EnterpriseServer
ReportingApplication
:ReportingGUI
ReportingIf
:ReportingLogic
ProductDispatcher
ProductDispatcherIf
MoveGoodsIf
EnterpriseQueryIf
PersistenceIf
StoreQueryIf
:Data
:Enterprise
:Persistence
:Store
*
:StoreServer
StoreApplication
:StoreGUI
StoreIf
:StoreLogic
ReportingApplication
:ReportingGUI
ReportingIf
:ReportingLogic
StoreQueryIf
PersistenceIf
EnterpriseQueryIf
:Data
:Store
:Persistence
:Enterprise

Static view

- Key abstractions
  - Component frame
    - Black-box view – no internal structure visible (type)
  - Component architecture
    - Glass-box view – first level subcomponents (instances) visible


Info
value : String
NamedEntity
name : String
Feature
value : String
info
1..*
0..*
behavior
0..*
communicationFeature
Version
version : String
prevVersion : String
version
InterfaceType
signature : String
interfaceType
<<enumeration>>
ConnectionType
normal
utility
entity
Property
type : String
property
0..*
0..*
property
VersionedEntity
locked : boolean
addInfo(name : String,value : String) : Info
Interface
communicationStyle : String
connectionType : ConnectionType
isCollection : boolean
+addFeature(name : String,value : String) : Feature
+addFactoryAnnotation(method : String,retInstance : Frame,retIface : Interface) : void
returnInterface
0..*
requiredInterface
0..*
providedInterface
annotation
0..*
Annotation
Frame
+addProvision(name : String,iface : InterfaceType,comStyle : String,conType : ConnectionType,isCol : boolean) : Interface
+addRequirement(name : String,iface : InterfaceType,comStyle : String,conType : ConnectionType,isCol : boolean) : Interface
+addProperty(name : String,type : String) : Property
+setTopLevel(topLevel : boolean) : void
+addBehavior(name : String,value : String) : Info
0..*
annotation
+returnInstance
as
MappedProperty
subcomponentPropertyName : String
0..*
mappedProperty
1..*
implements
0..1
instantiatesFrame
<<Singleton>>
TopLevel
<<Singleton>>
Factory
method : String
SubcomponentInstance
0..*
subcomponent
Architecture
implementation : String
addSubcomponentByFrame(name : String,frame : Frame) : SubcomponentInstance
addSubcomponentByArch(name : String,arch : Architecture,frame : Frame) : SubcomponentInstance
addConnection() : Connection
addProperty(name : String,type : String) : Property
addMappedProperty(name : String,type : String,subcompPropName : String,subc : SubcomponentInstance) : MappedProperty
instantiatesArchitecture
0..1
instance
subcomponent
connection
0..*
Connection
addComponentEndpoint(ifaceName : String) : ComponentInterfaceEndpoint
addSubcomponentEndpoint(ifaceName : String,sub : SubcomponentInstance) : SubcomponentInterfaceEndpoint
endpoint
1..*
ConnectionEndpoint
ComponentInterfaceEndpoint
interfaceName : String
SubcomponentInterfaceEndpoint
interfaceName : String

Behavior view I. – an example




```
component LightDisplay {

  types {
    STATES = { LIGHT_ENABLED,
    LIGHT_DISABLED }
  }

  vars {
    STATES state = LIGHT_ENABLED
  }
```

```
behavior {
?LDispCtrlEventHandlerIf.onEvent(STATES e) {
  switch (e) {
    LIGHT_ENABLED:
          {state <- LIGHT_ENABLED }
    LIGHT_DISABLED:
          {state <- LIGHT_DISABLED }
  }
 }*
}
}
```

Behavior view II.

BP (SOFA, Fractal):

```
(
?LDispCtrlEventHandlerIf.
    onEvent_ENABLE
+
?LDispCtrlEventHandlerIf.
    onEvent_DISABLE
)*
```

EBP (SOFA 2.0):

```
component LightDisplay {
  types {
    STATES = { LIGHT_ENABLED, LIGHT_DISABLED }
  }
  vars {
    STATES state = LIGHT_ENABLED
  }
  behavior {
    ?LDispCtrlEventHandlerIf.onEvent(STATES e) {
      switch (e) {
        LIGHT_ENABLED :
                    {state <- LIGHT_ENABLED }
        LIGHT_DISABLED :
                    {state <- LIGHT_DISABLED }
      }
    }*
  }
}
```

Recall: original BP

- Behavior protocol
  - Expression describing the behavior of a software component
    - Infinite set of finite event traces

- 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 + τ)
    indicating communication errors
    no activity (deadlock)
    bad activity (! cannot be responded)

- Syntactic abbreviations (to express method calls)
  - **`?i.m`** = **`?i.m`**↑ ; **`!i.m`**↓
  - **`?i.m{`***prot*`}` = **`?i.m`**↑ ; *prot* ; **`!i.m`**↓

Recall: original BP

- Behavior protocol
  - Expression describing the behavior of a software component
    - Infinite set of finite event traces

- Events:
  - Emitting a method call request: `!interface.method(arg_list)↑`
  - Accepting a method call request: `?interface.method(par_list)↑`
  - Emitting a method call response: `!interface.method↓`
  - Accepting a method call response: `?interface.method↓`
  - **Assignment to a variable** `var ← value`
  - **Multisynchronization event** `@sync`

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

  = parallel composition (interleaving + τ)
  indicating communication errors
  no activity (deadlock)
  bad activity (! cannot be responded)

- Syntactic abbreviations (to express method calls)
  - `?i.m = ?i.m↑ ; !i.m↓`
  - `?i.m{`*prot*`} = ?i.m↑ ;` *prot* `; !i.m↓`

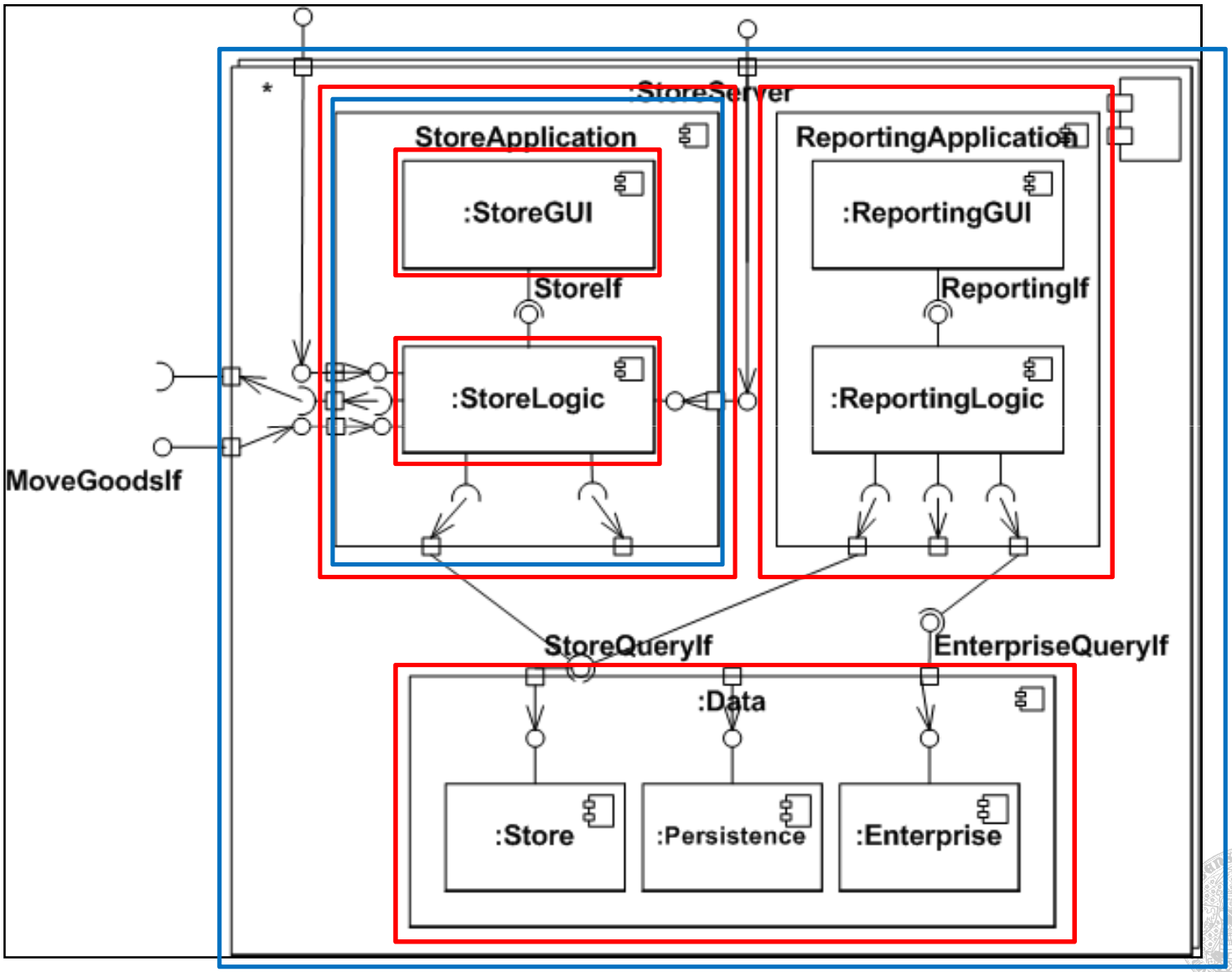
Behavior view III - features

- Extended Behavior Protocol (EBP)
  - Expression defining the desired finite sequences (traces) of
    - Atomic Events
      - method call request and response
      - assignment to a local variable, multisynchro event.
  - BP like: Behavior spec. of communicating components
    - composed via (extended) consent operator
    - detection of composition errors
      - Bad activity – a request cannot be accepted
      - No activity – deadlock
  - Models component modes
    - Method parameters and component-local variables of enumeration types
  - Allow synchronization of events from more than two EBPs – multisynchronization
    - Not specifically used in CoCoME

Behavior view III - compliance

- Two compatibility relations are verified
  - Horizontal compliance ~ absence of communication errors within the consent composition of EBP of all first-level subcomponents of a composite components
  - Vertical compliance ~ absence of communication errors in composition of the inverted frame and architecture protocols
  - Verified by a tool chain
    - ebp2prom – transforming EBP into Promela
    - Spin – model checking the Promela specification

Behavior view III (example)



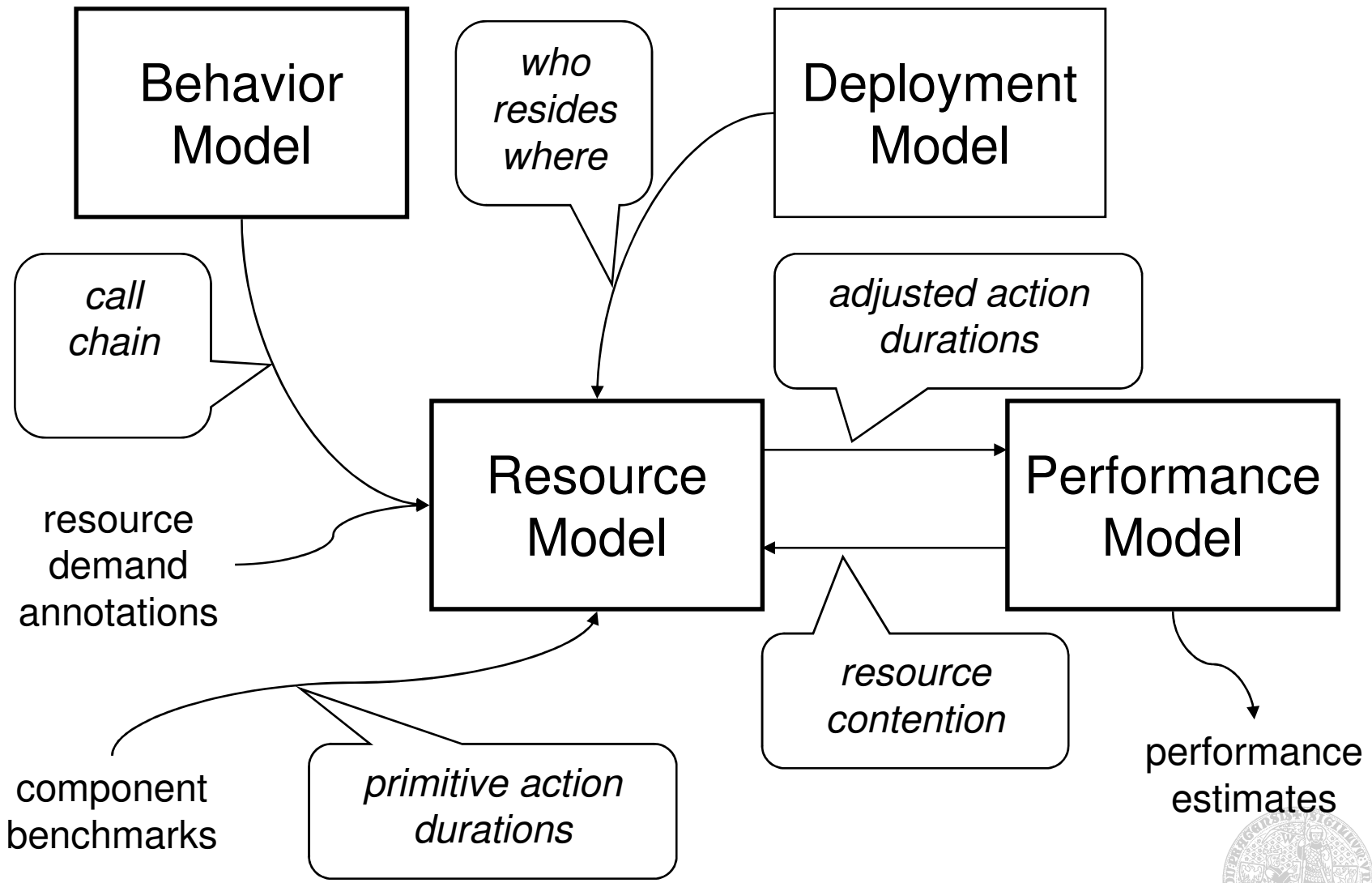
EBP Benefits

- EBP are a concise specification platform for component behavior:
  - EBP spec integrates information from
    - Code
    - Component diagram (UML)
    - Use cases
    - Sequence diagrams (UML)
      - These not that useful (all info in the UC and code)
  - EBP enable:
    - Detecting composition errors at design time
    - Verification against code
      - Work in progress
    - Listing of all traces corresponding to a single request
      - Work in progress
    - Verification whether a use case is actually implemented in the code
      - Work in progress
  - EBP spec respect the component hierarchy
    - Sequence diagrams do not

Deployment view

- Distributed runtime environment (SOFAnode)
  - a repository
    - Storage for metadata and code
  - deployment docks
    - Container for components' instantiation and run
- Component lifecycle
  1. Creation and upload to the repository
  2. Assembly of components
  3. Deployment and launching
- Deployment spec: deployment plan

Performance view (Overview)



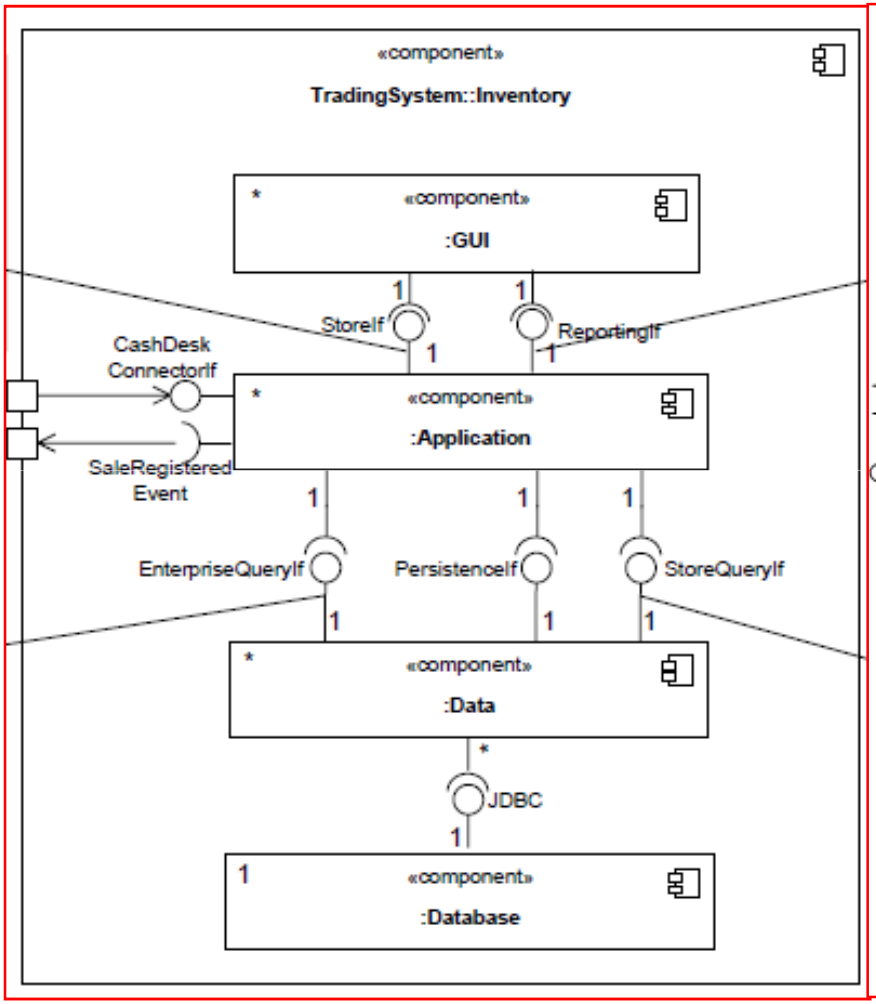
CoCoME in SOFA 2.0

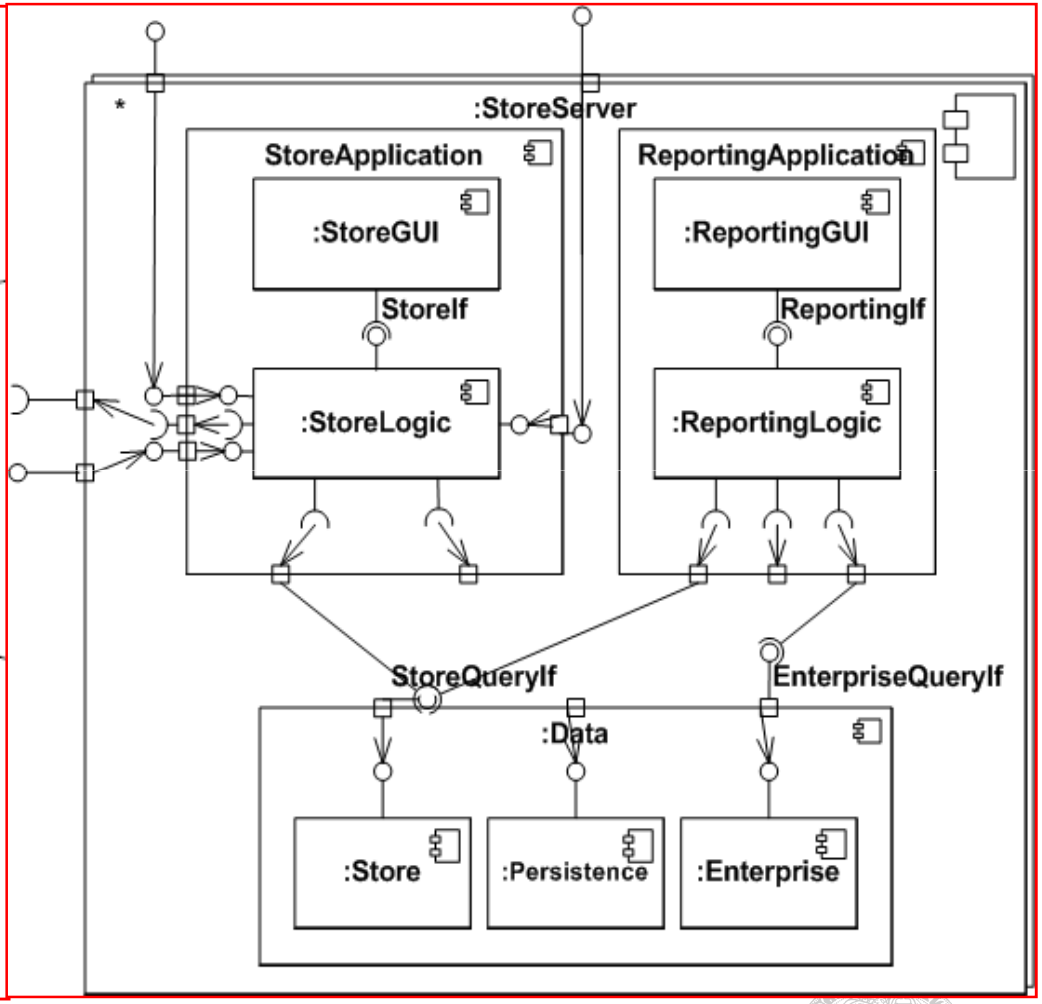
- Focused on
  - Modeling behavior:
    - Extended Behavior Protocols
      - Verification of vertical and horizontal compliance
      - Input to performance modeling
  - Modeling performance:
    - Layered Queuing Networks
    - Resource model
      - Prediction of components' resource usage attributes

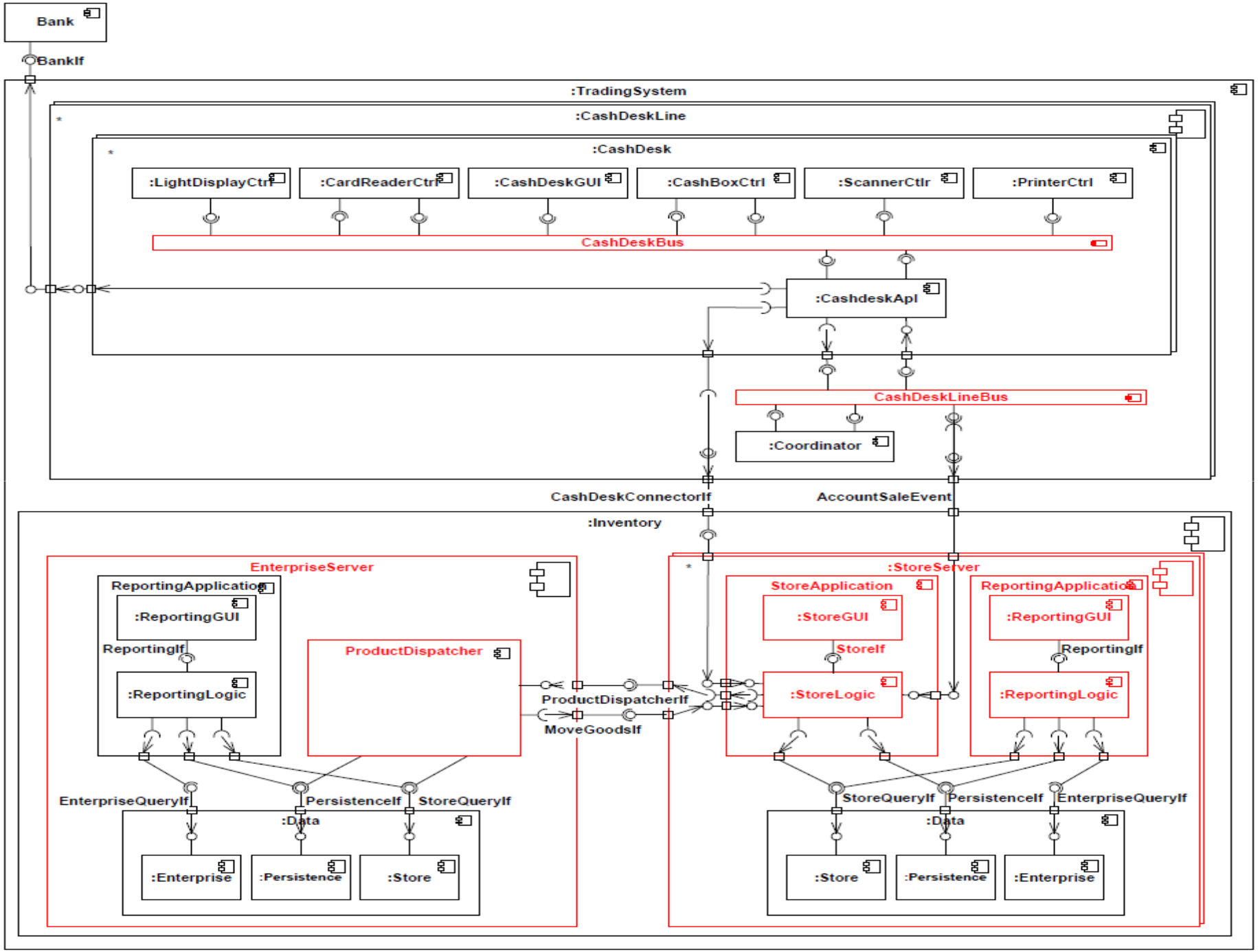
SOFA 2.0: Static view ~ UML assignment

- Complies ~ with UML assignment
  - External hierarchical bus replaced by independent buses inside components
    - Better reflects the orthogonal activities of C*ashDeskLine* and *Coordinator*
    - The number of *CashDesk*s and *CashDeskBus*es is equal
      - Not visible from UML spec (*)
  - Added: Interface and bindings to support UC8
    - Adding *EnterpriseServer* and *StoreServer*
  - Modification of *Inventory*
    - *Application* and *GUI* replaced by *StoreApplication* and *ReportingApplication*
      - Better captures independence of *Store* and *Reporting*

SOFA 2.0: Static view ~ UML assignment





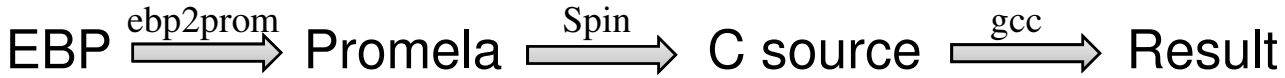

Bank
BankIf
:TradingSystem
:CashDeskLine
*
:CashDesk
*
:LightDisplayCtrl
:CardReaderCtrl
:CashDeskGUI
:CashBoxCtrl
:ScannerCtlr
:PrinterCtrl
CashDeskBus
:CashdeskApl
CashDeskLineBus
:Coordinator
CashDeskConnectorIf
AccountSaleEvent
:Inventory
EnterpriseServer
ReportingApplication
:ReportingGUI
ReportingIf
:ReportingLogic
ProductDispatcher
ProductDispatcherIf
MoveGoodsIf
EnterpriseQueryIf
PersistenceIf
StoreQueryIf
:Data
:Enterprise
:Persistence
:Store
*
:StoreServer
StoreApplication
:StoreGUI
StoreIf
:StoreLogic
ReportingApplication
:ReportingGUI
ReportingIf
:ReportingLogic
StoreQueryIf
PersistenceIf
EnterpriseQueryIf
:Data
:Store
:Persistence
:Enterprise

SOFA 2.0: Behavior view

- Modeled in EBP
  - Based on the provided reference implementation, UCs, and sequence diagrams
    - Inconsistencies between UCs adjusted according to the reference implementation
  - Behavior of each component: its frame protocol
    - Integrates effect of a set of sequence diagrams
      - All possible interplays of calls (accepted and issued) captured
  - Actors (*Customer*, *Cashier*, …) modeled indirectly
    - inside GUI components
  - Multiplicity
    - Two *CashDesk*s inside each *CashDeskLine*
    - Two *CashDeskLine*s inside the *TradingSystem*

Results – Behavior modeling I.

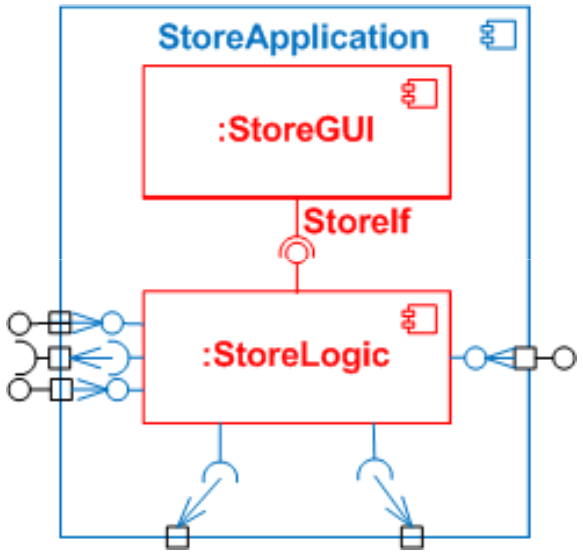
- Specification in EBP takes about 1500 LOC (42kB)
  - Verifying horizontal and vertical compliance:



- Tools:
  - ebp2prom – transformation EBP → Promela
    - A frame protocol transformed into a finite automaton
    - Compliance checking → assertion checking
  - Spin – verification of the Promela model
    - Error trace provided in EBP
- Running on
  - PC 2x Intel Core2 Duo (dual core), 4GB RAM, Gentoo Linux

Results – Behavior modeling II.

- Demo – verification of *StoreApplication*



Results – Behavior modeling III.

- Total time = ebp2prom + Spin + gcc time + verifier

Component	ebp2prom	Verifier	Total	States
CashDesk	41.5	46.1	97.1	3,335,950
CashDeskLine	59.6	1.0	71.2	3,912
StoreApplication	37.5	3.2	50.1	378,466
Data	159.8	9.8	203.6	1,119,740
ReportingApplication	0.3	1.0	1.6	39
StoreServer	154.3	15.3	19[illegible]	2,0[illegible]
EnterpriseServer	151.9	1.0	1[illegible]	2[illegible]
Inventory	0.5	1.0	1[illegible]	3[illegible]
TradingSystem	54.0	1.4	6[illegible]4	[illegible]79
Total time	659.4	79.8	869.1	[illegible]1,743

Performance view

- Goals
  - Estimate performance during design
    - Very rough take at absolute numbers
    - Good approximation of scalability
  - Model implicitly shared resources
- Issues
  - Complexity of performance models
  - Presence of black box components
- Solution
  - Multiple sources of information
    - Behavior model tracks resource demand
    - Deployment model tracks component placement
    - Resource model provides resource usage information
    - Component benchmarks provide action duration information
  - Iterative model solution

SOFA 2.0: Performance view I.

- Assumptions
  - Cash desk performance not an issue
  - Central server performance vital
    - No scalability support in architecture
    - Sustainable peak performance of interest

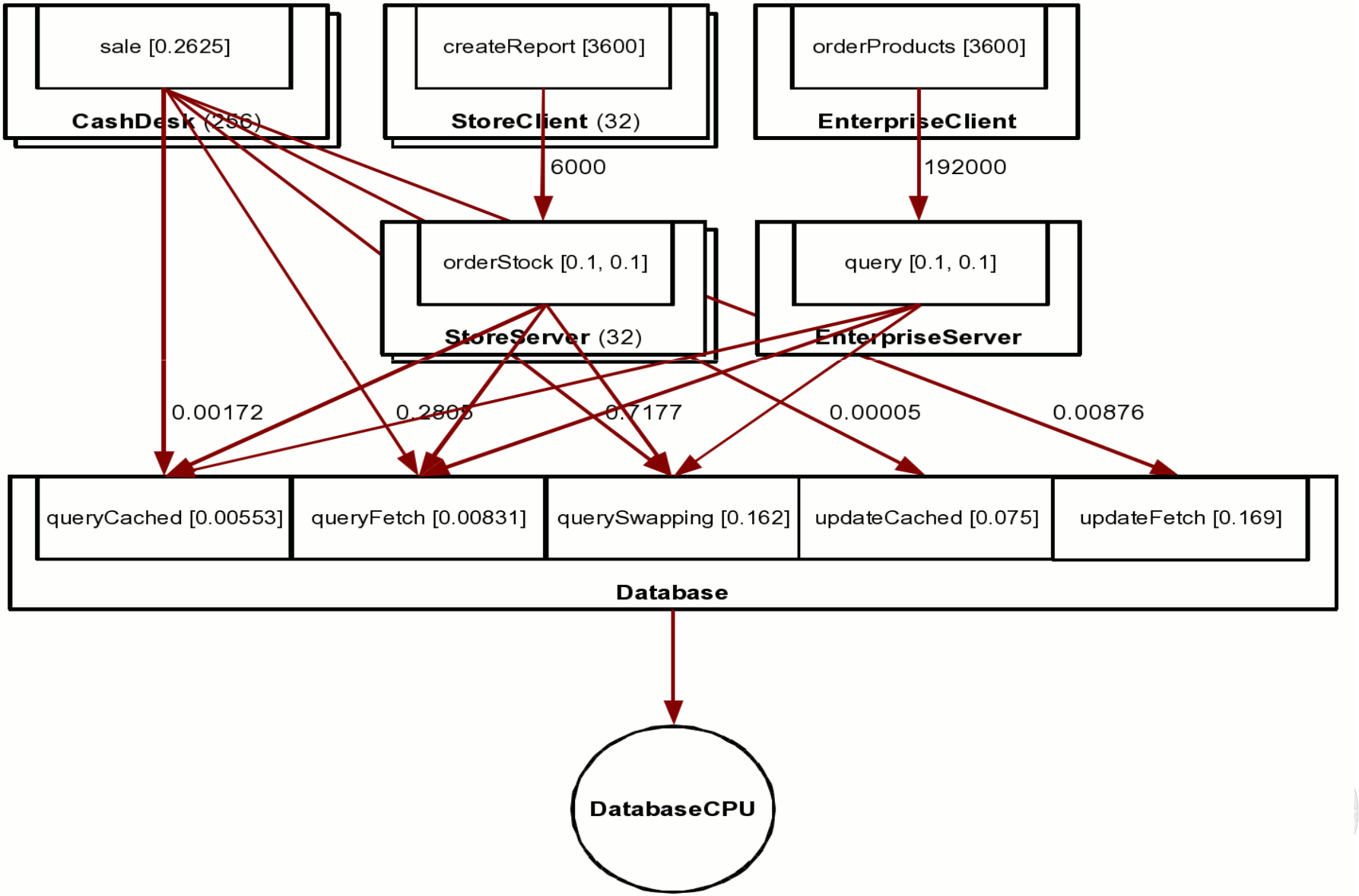
- Component Benchmarks
  - Database the major black box component
  - Query duration for selected queries
  - Scalability investigation
    - Database size irrelevant
    - Cache behavior significant
    - Memory consumption significant
  - Resource consumption investigation

SOFA 2.0: Performance view II.

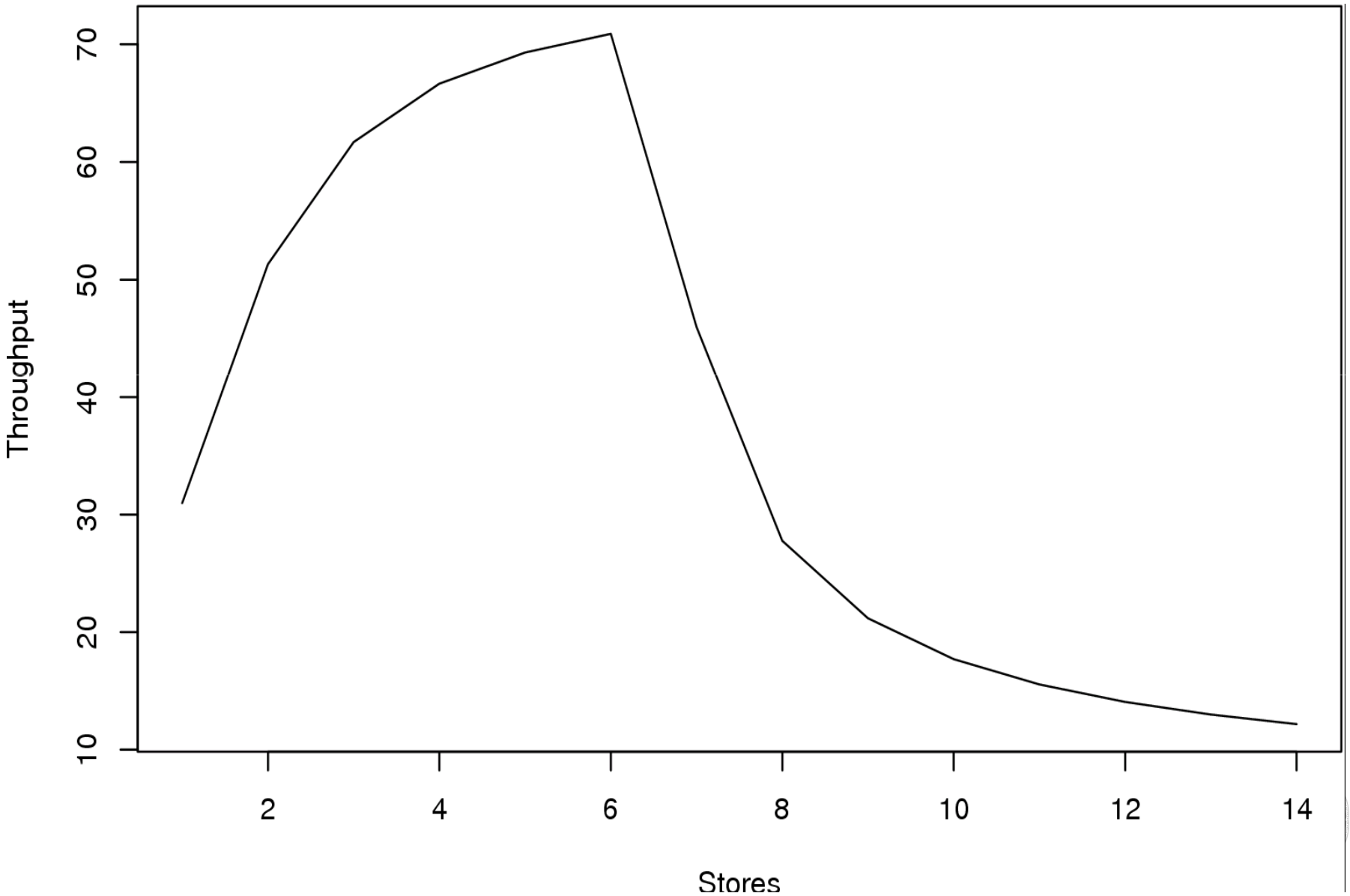
- Resource Model
  - Database cache usage
    - Ratio of cache to database size
    - Requires formula for database size
  - System memory usage
    - Store instance related consumption
    - Query processing related consumption
  - Output
    - Query duration for database queries
    - Adjusted based on cache usage and memory usage

- Performance Model
  - Layered queuing network
  - Average durations considered
  - Multiple customers aggregated
  - Output
    - Throughput and roundtrip estimates
    - Concurrency for database queries

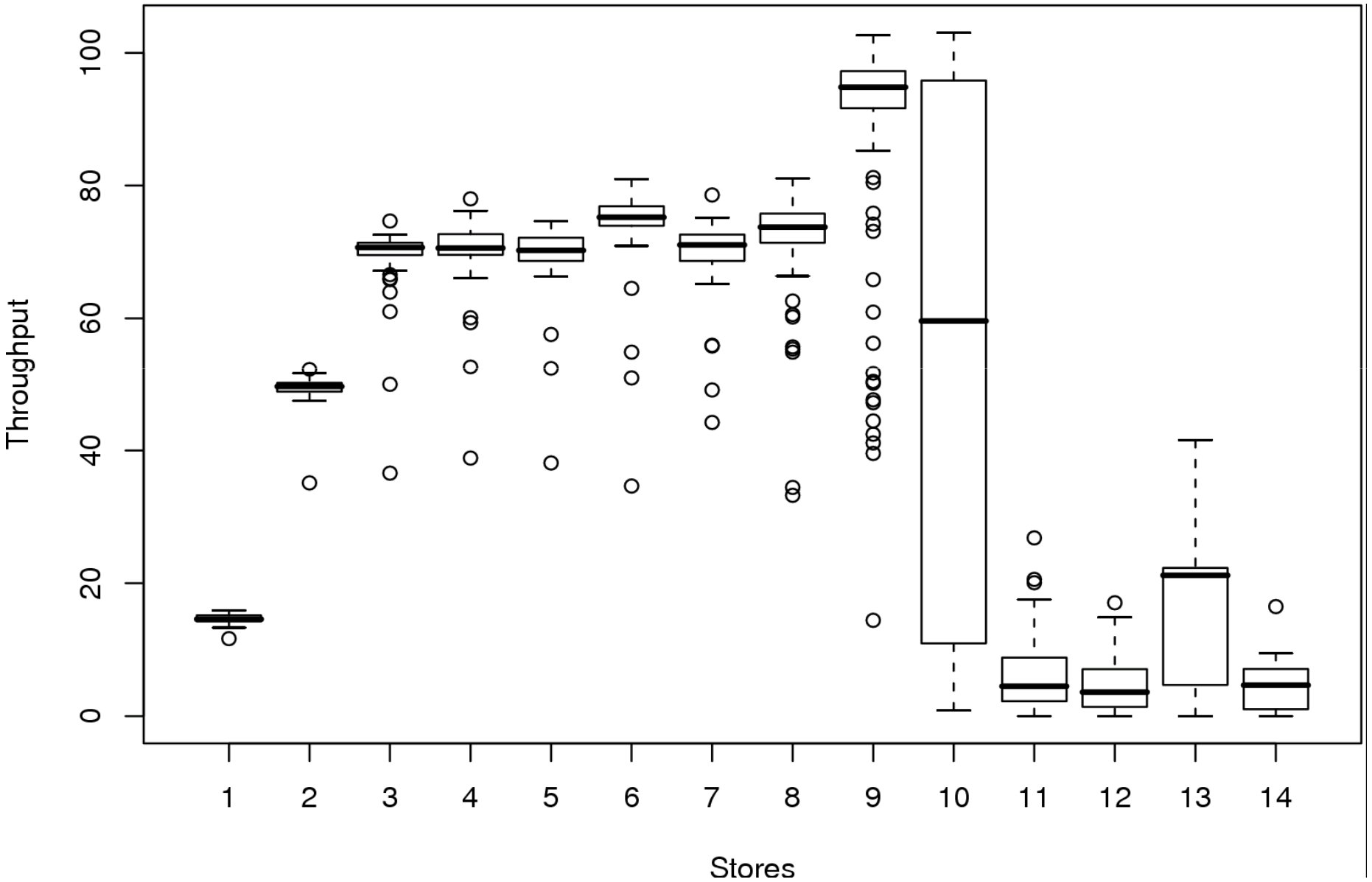
LQN model


sale [0.2625]
CashDesk (256)
createReport [3600]
StoreClient (32)
orderProducts [3600]
EnterpriseClient
6000
192000
orderStock [0.1, 0.1]
StoreServer (32)
query [0.1, 0.1]
EnterpriseServer
0.00172
0.2805
0.7177
0.00005
0.00876
queryCached [0.00553]
queryFetch [0.00831]
querySwapping [0.162]
updateCached [0.075]
updateFetch [0.169]
Database
DatabaseCPU

Results of modeling



Measured results



Evaluation I.

- EBP are a concise specification platform for component behavior:
  - EBP spec integrates information from
    - Use cases
    - Code
    - Component diagram (UML)
    - Sequence diagrams (UML)
      - These not that useful (all info in the UC and code)
  - EBP enable:
    - Detecting composition errors at design time
    - Verification against code
      - Work in progress
    - Listing of all traces corresponding to a single request
      - Work in progress
    - Verification whether a use case is actually implemented in the code
      - Work in progress
  - EBP spec respect the component hierarchy
    - Sequence diagrams do not

Evaluation II.

- EBP vs. BP comparison
  - EBP – more detailed
    - modeling of method (some) parameters and component states (modes)
  - EBP spec more readable and easier to "debug"
  - Both verified in reasonable time (minutes)
- Compared to CoCoME in Fractal
  - Specification in EBP shorter (1500 vs. 2700 LOC) and more readable

Conclusion

- Behavior:
  - Both vertical and horizontal compliance verified in reasonable time
- Performance
  - Simple models with reasonably precise output
- *Available at*

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

Future Work

- Behavior view
  - Checking of EBP spec against Java code
    - Taking advantage of data in spec
  - Support for sessions
  - Support for other properties
    - E.g. enhancing data relations
- Performance view
  - Additional behavioral information needed to generate model automatically
  - Potential for automating resource modeling
    - Component benchmarking
    - Knowledge of typical resources