Linking Programs to Architectures: An Object-Oriented Hierarchical Software Model based on Boxes

Jan Schäfer Markus Reitz Jean-Marie Gaillourdet
Arnd Poetzsch-Heffter

University of Kaiserslautern, Germany
Department of Computer Science
Software Technology Group
Prof. Dr. Arnd Poetzsch-Heffter

CoCoME Dagstuhl Seminar
August 2, 2007

Expertise

- Strong Experience
  - Design and Implementation of Programming Languages
  - Specification and Verification of OOLs
  - Compiler Construction, Verification, and Optimization
  - Component-Oriented Development
  - Theorem Provers
  - Type Systems
- Gaining Experience
  - Adaptive and Reconfigurable Systems
  - Concurrent and Distributed Systems
  - Software Architectures

Outline

Our Approach for CoCoME

Our Approach for CoCoME

Common Component Modeling Example

Our Approach for CoCoME

Common Component Modeling Example

Our Approach for CoCoME

Common Component Modeling Example

Why do we need components?

Why do we need components?

To enable reuse and substitution of software artifacts

Why do we need components?

To enable reuse and substitution of software artifacts

... and modular verification

What are reuse and substitution?

What are reuse and substitution?

Reuse means to use one component in multiple environments.

What are reuse and substitution?

Reuse means to use one component in multiple environments.

Substitution means to replace a component by another component in one environment.

What are reuse and substitution?

Reuse means to use one component in multiple environments.

one component – multiple environments

Substitution means to replace a component by another component in one environment.

multiple components – one environment

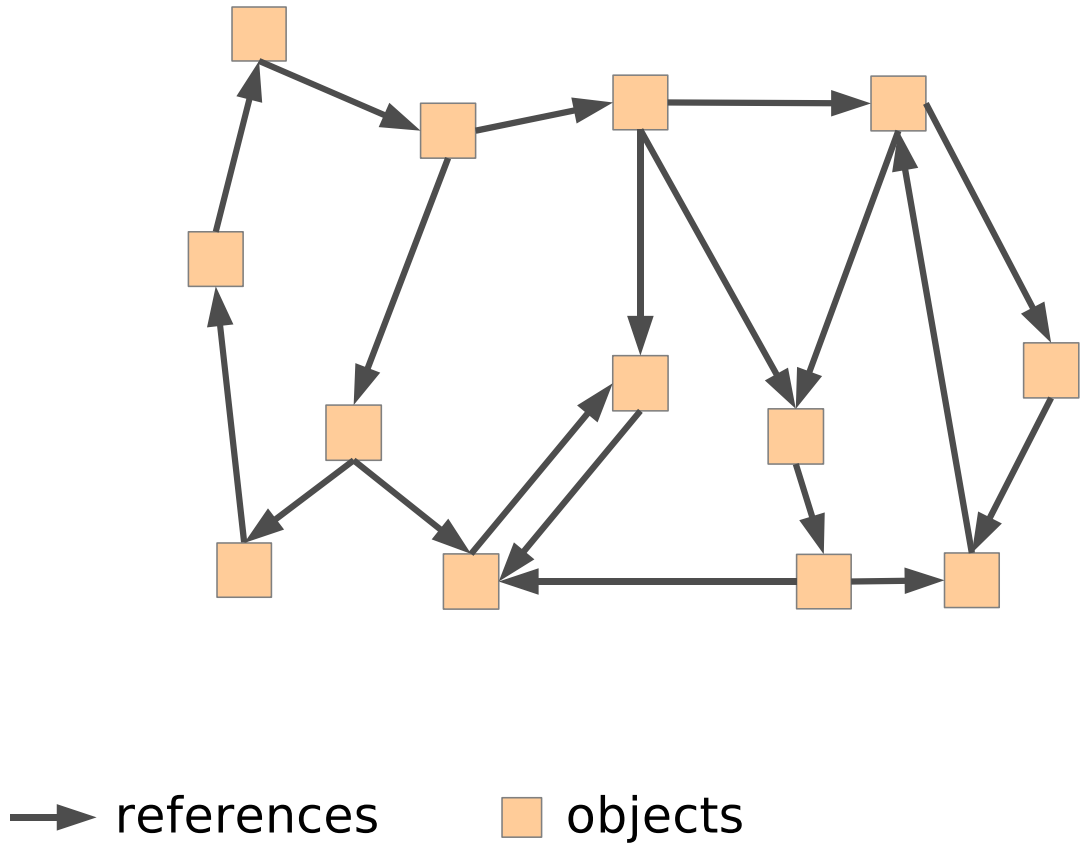
When are reuse and substitution possible?

When are reuse and substitution possible?

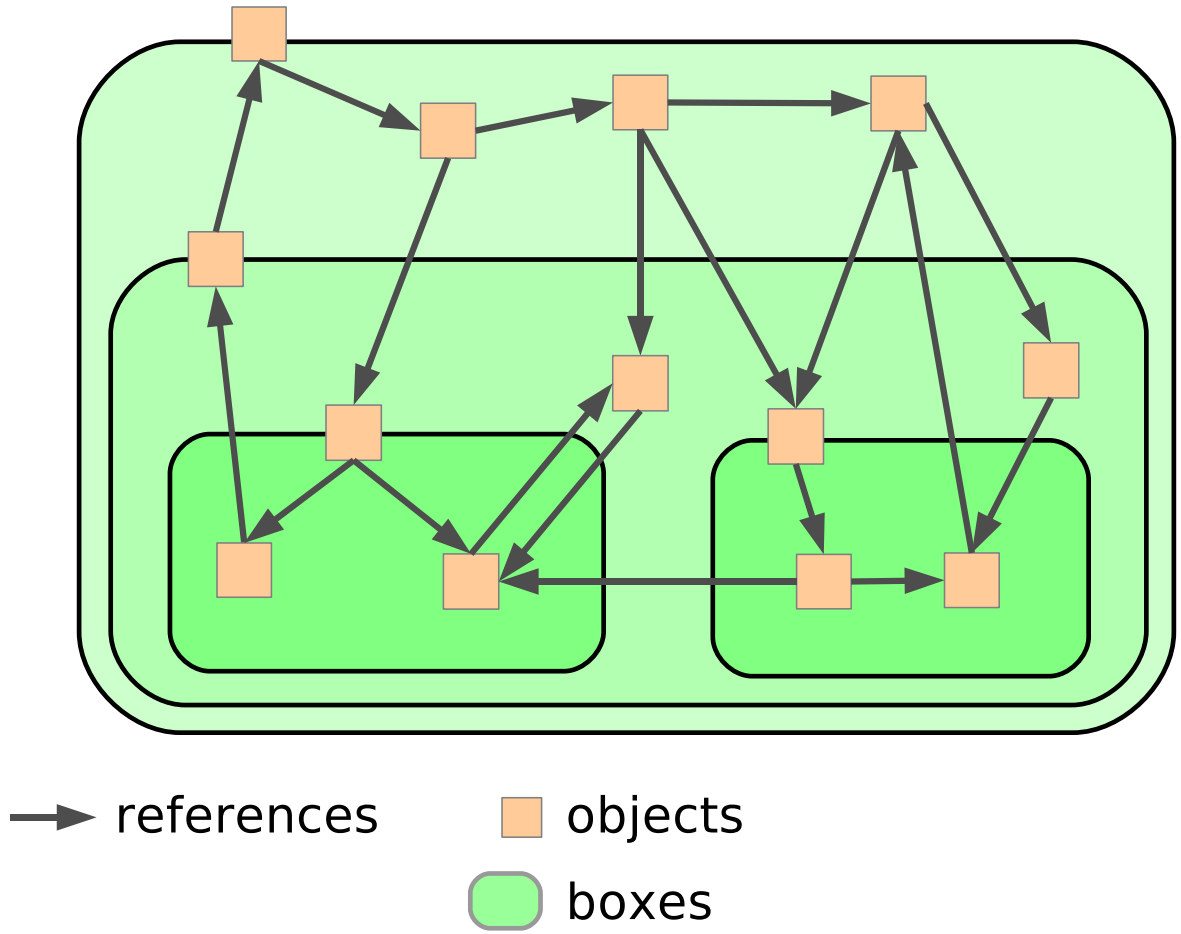
We have to understand

- compatibility of interfaces
- compatibility of behaviors
- what constitutes a component
  - statically
  - dynamically

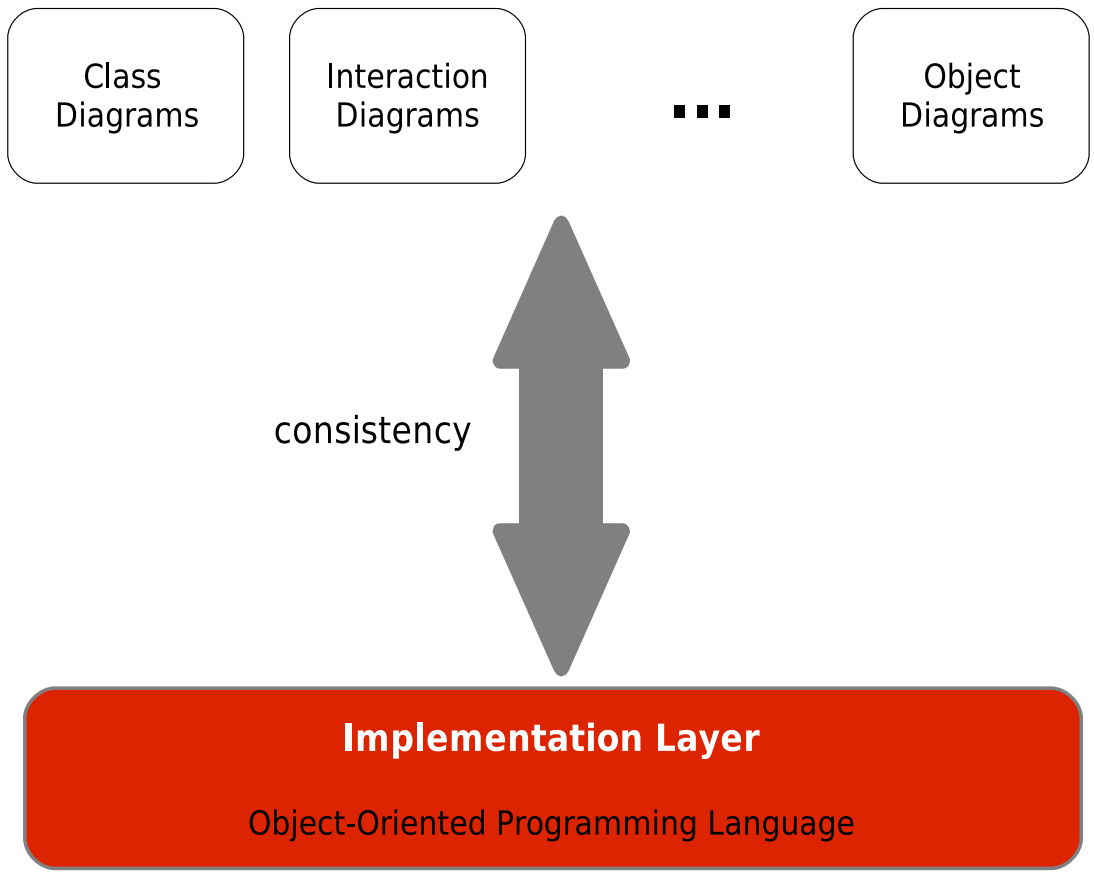
Unstructured Object-Heap



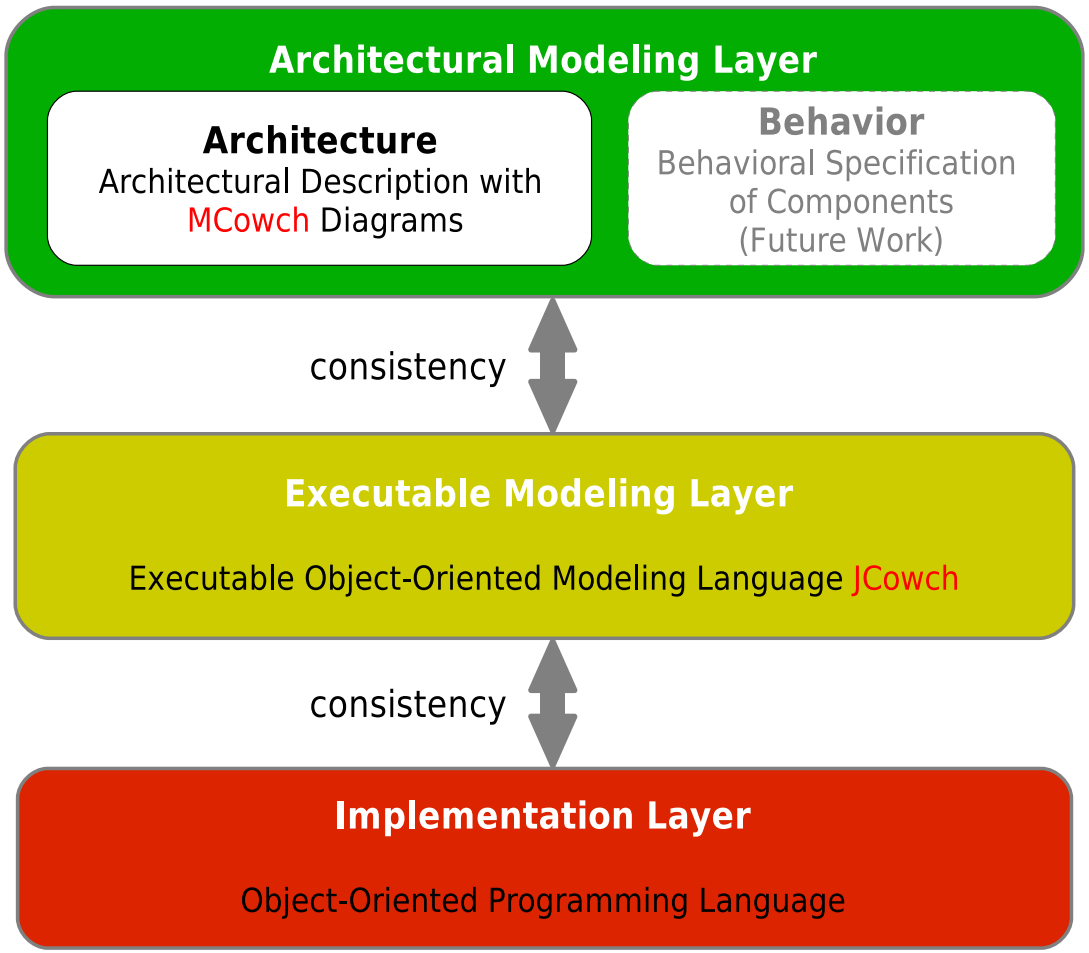
Heap structured with Boxes



Standard Approach: UML



Our Approach: Cowch

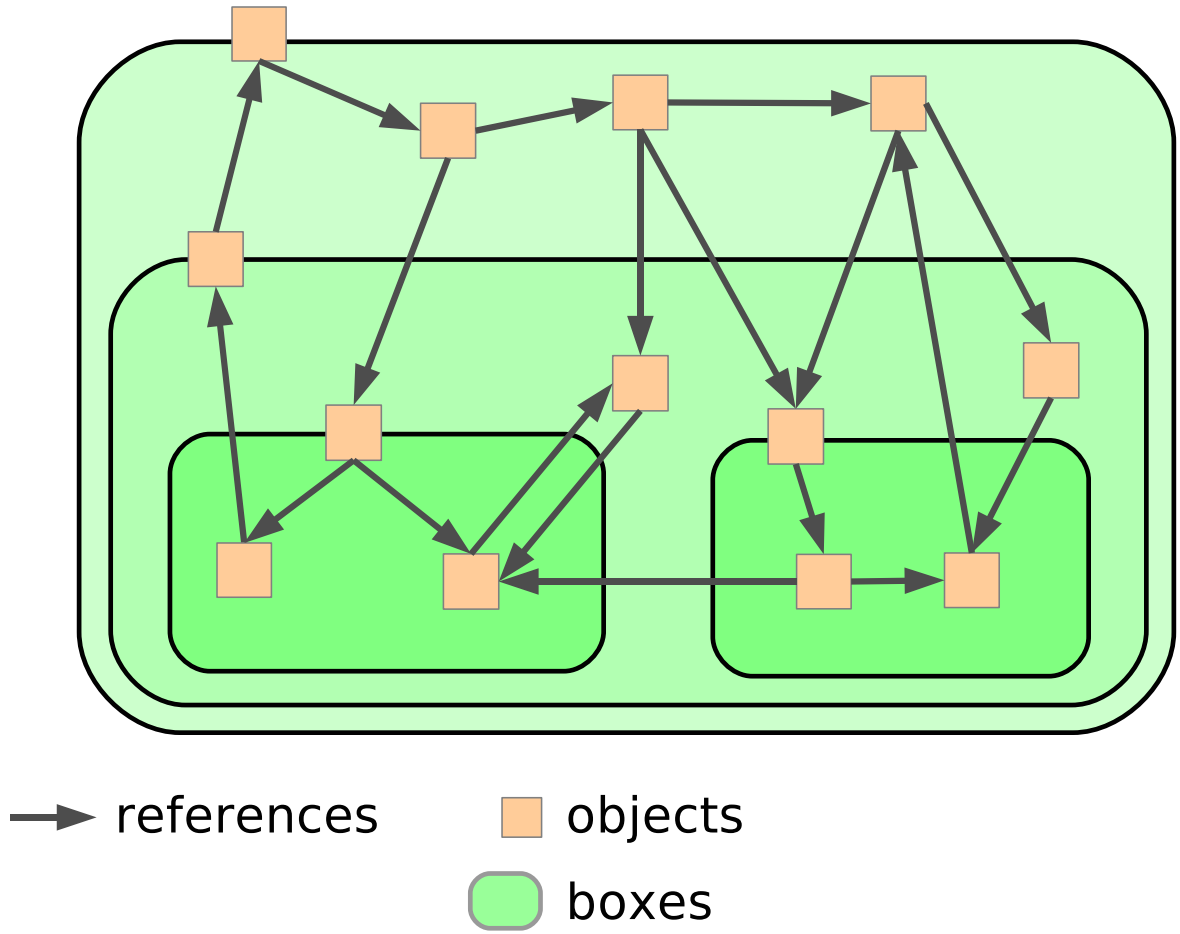


Executable Modeling Layer

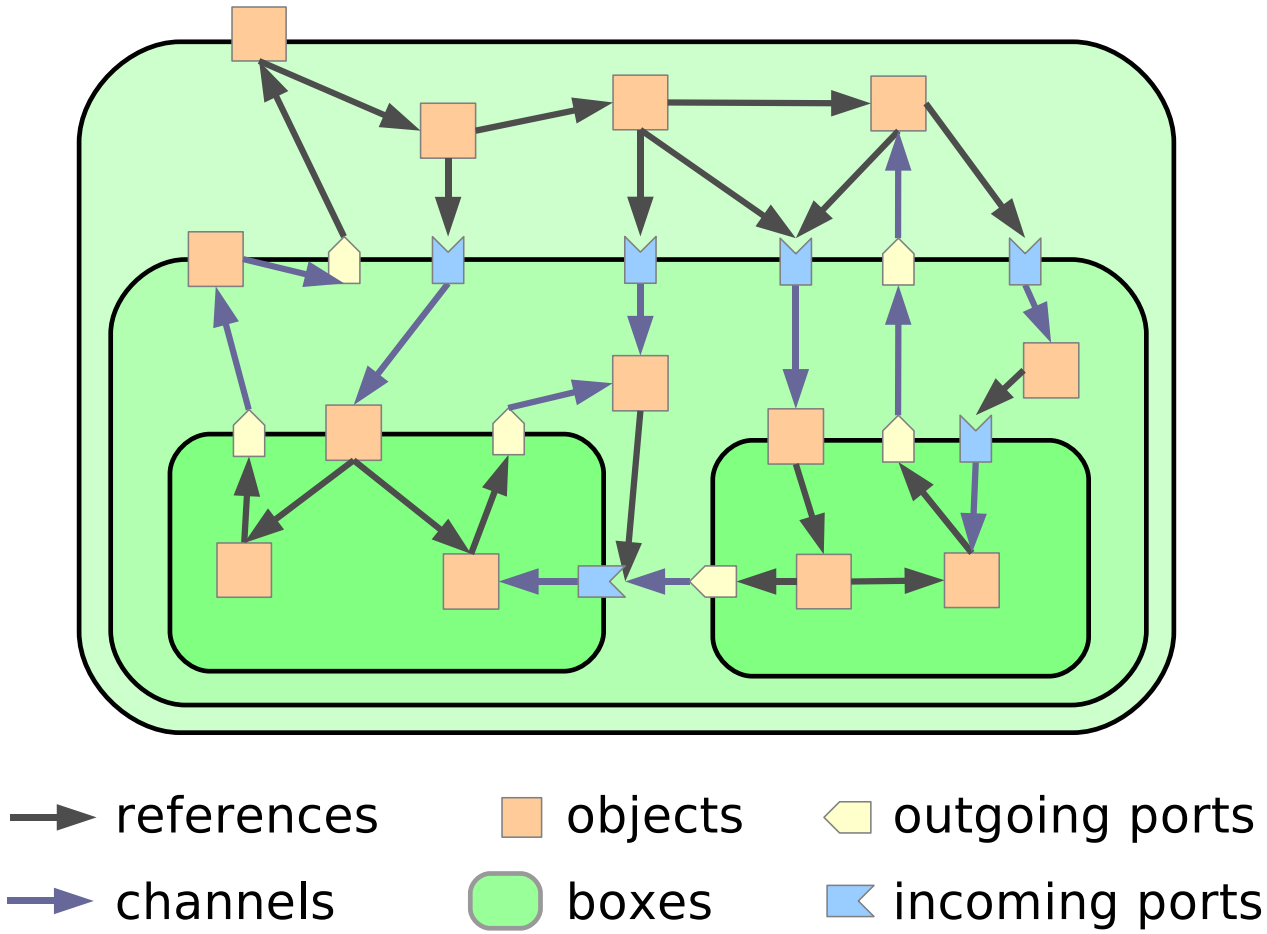
JCowch

- Executable Language
- Object-Oriented
- Classes and Interfaces
- Components and Boxes
- Ports and Channels
- High-Level Concurrency Model

Heap structured with Boxes



Heap structured with Boxes, Ports, and Channels



Ports and Channels

Ports

- Explicitly named interfaces at the boundary
- Outgoing or incoming
- Typed by interfaces
- Referenceable like ordinary objects

Channels

- Connect ports/ports and ports/objects
- Explicit creation/destruction
- Method-level channels possible

Concurrency

Based on the Join Calculus

- Asynchronous methods
- Join Patterns
- No explicit thread creation
- No explicit locks

JCowch Example

Producer–Consumer

```
box class Producer {
  outport StringBuffer buffer;
  async run() {
    while (true) {
      String s = "Hello World";
      buffer.put(s);
    }
  }
}

box class Consumer {
  outport StringBuffer buffer;
  async run() {
    while (true) {
      String s = buffer.get();
      System.out.println(s);
    }
  }
}
```

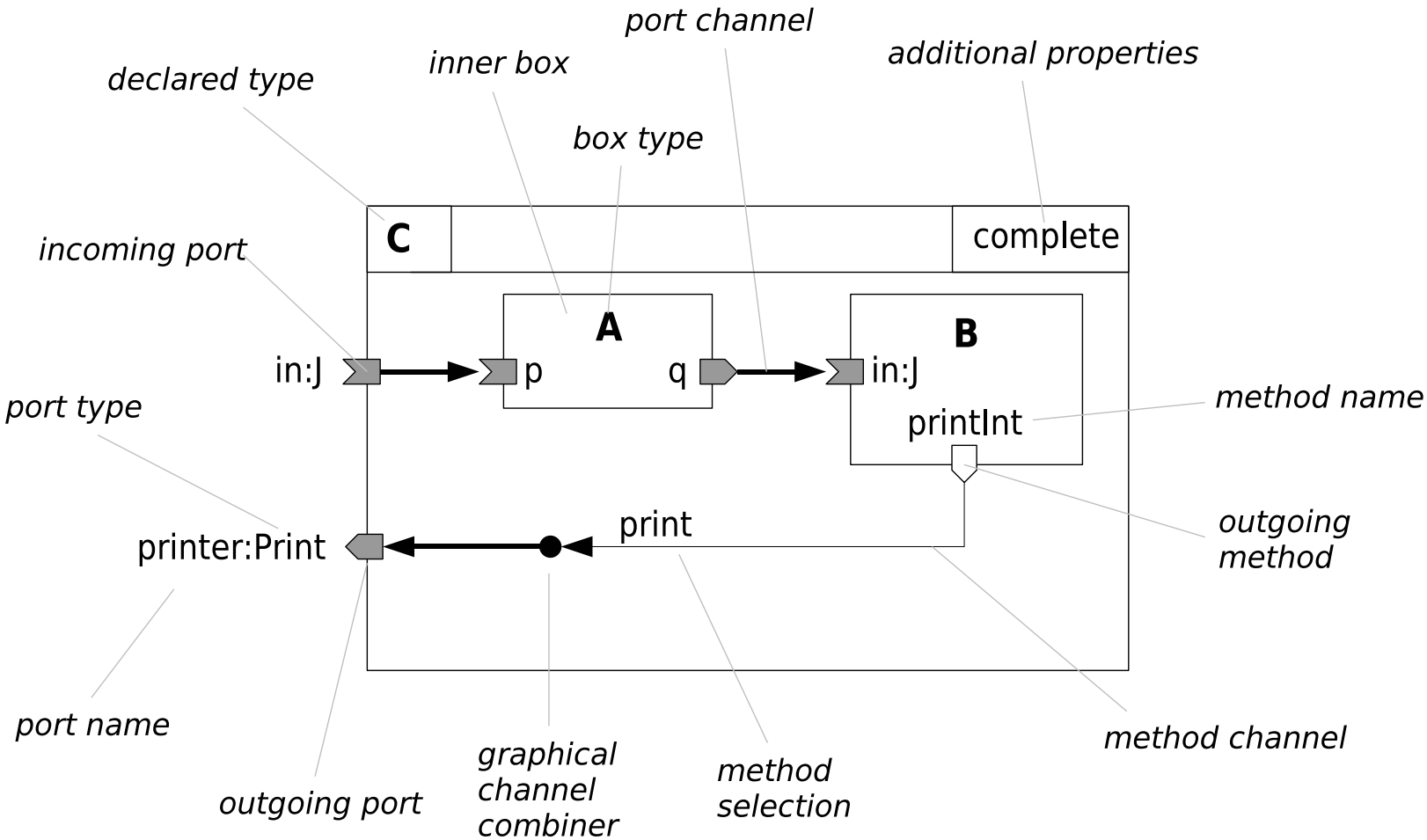
```
interface StringBuffer {
  async put(String s);
  String get();
}

box class StringBufferImpl implements StringBuffer {
  String get() & put(String s) { return s; }
}

box class Main {
  void main() {
    StringBuffer buff = new StringBufferImpl();
    Producer prod = new Producer();
    Consumer cons = new Consumer();
    connect prod.buffer to buff;
    connect cons.buffer to buff;
    prod.run(); cons.run();
  }
}
```

Architectural Modeling Layer

MCowch

- Graphical non-executable language
- Describes the statically determined box structure
  - Hierarchy
  - Channels
- Semantic-based consistency relation to executable modeling layer
- In general underspecified
  - Also fully specified models supported

MCowch Syntax



Additional Properties

Complete Diagrams

- Fully specified models
- Architecture model $\widehat{=}$ Executable model
- Allows code generation

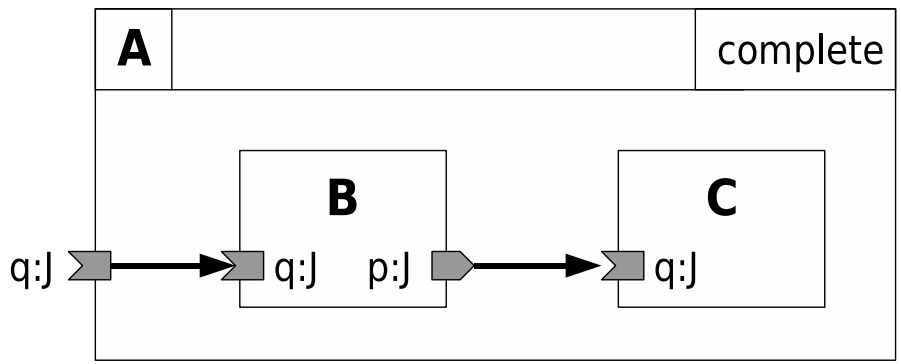
Active/Reactive Boxes

- Active Boxes
  - May spontaneously emit messages
- Reactive Boxes
  - Require external stimulation

Consistency Relation between MCowch and JCowch

Example 1/2

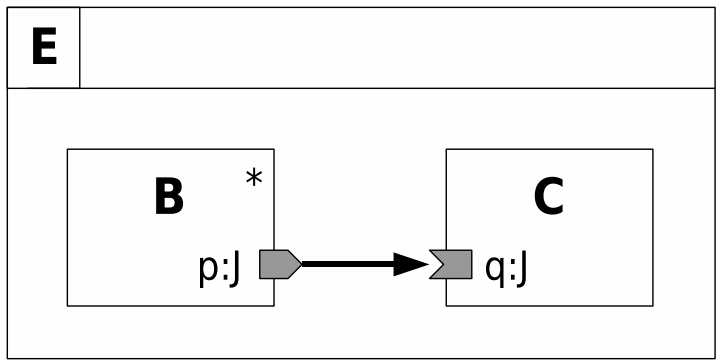
```
box class A {
  inport J q;
  B b; C c;
  A() {
    b = new B();
    c = new C();
    connect this.q to b.q;
    connect b.p to c.q;
  }
}
```



Consistency Relation between MCowch and JCowch

Example 2/2

```
box class E {
  C c;
  B[] b;
  E(int numB) {
    c = new C();
    b = new B[numB];
    for (int i=0; i <numB; i++) {
      b[i] = new B();
      connect b[i].p to c.q;
    }
  }
}
```



Summary (1)

Notion of a component

Which allows to define

- the constituting elements
- the interface
- the behavior

$\Rightarrow$ safe reuse and substitution come into reach

Modeling approach

- Two-Layered approach
- Consistency relation

Summary (2)

Scope

Distributed concurrent object-oriented systems

Focus

Structural and behavioral aspects

Our CoCoME Model

The Modeled Part of CoCoME

We provide

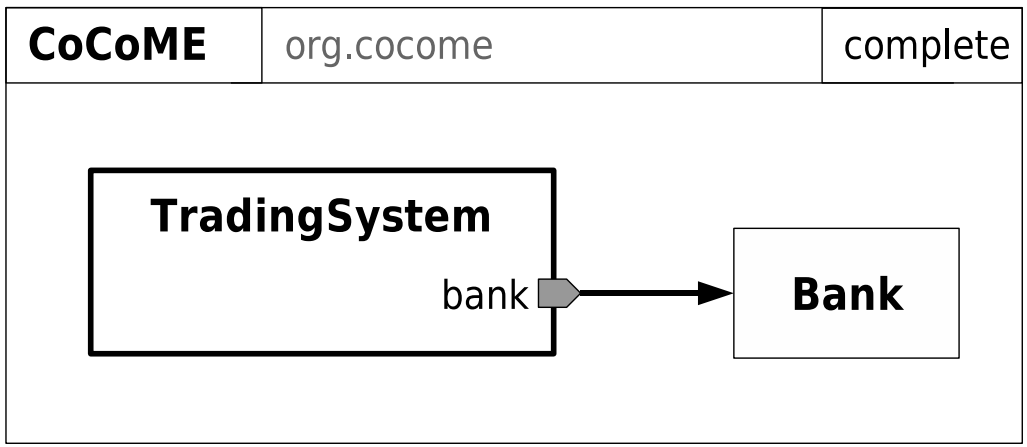
- Architectural Model
  - Most parts
    - No Inventory
    - No GUI
- Executable Model
  - Only exemplarily
    - Coordinator Component

We abstracted from

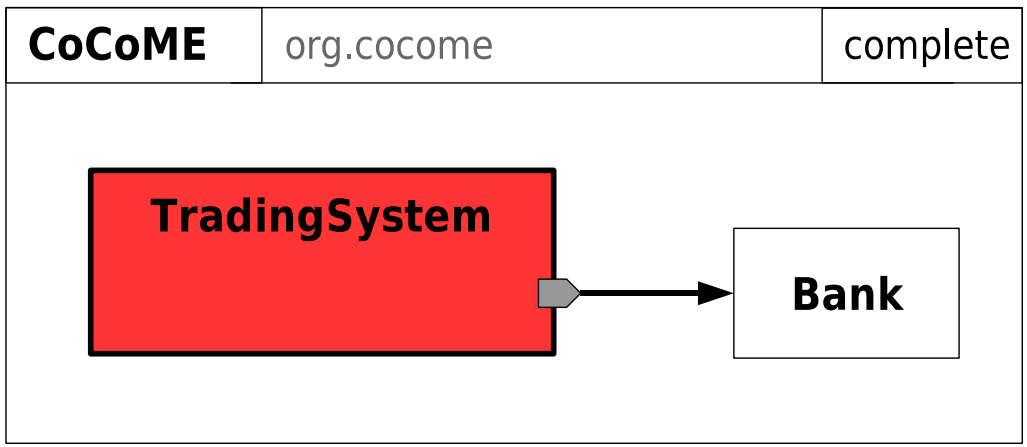
- Technologies
  - Hibernate, RMI, ActiveMQ, ...

Architectural Model

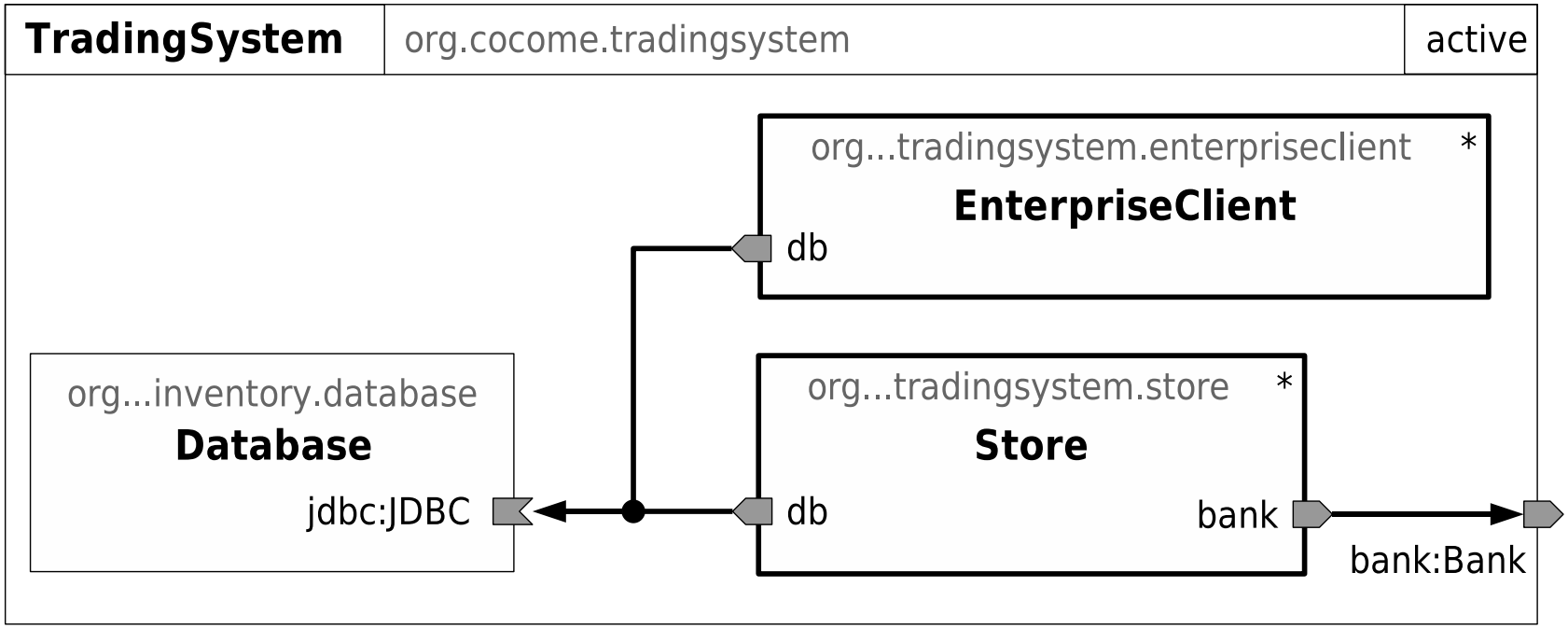
Top-Level View

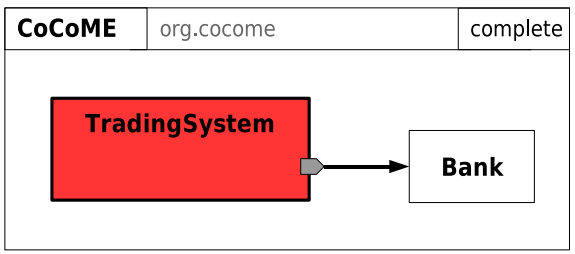


Top-Level View

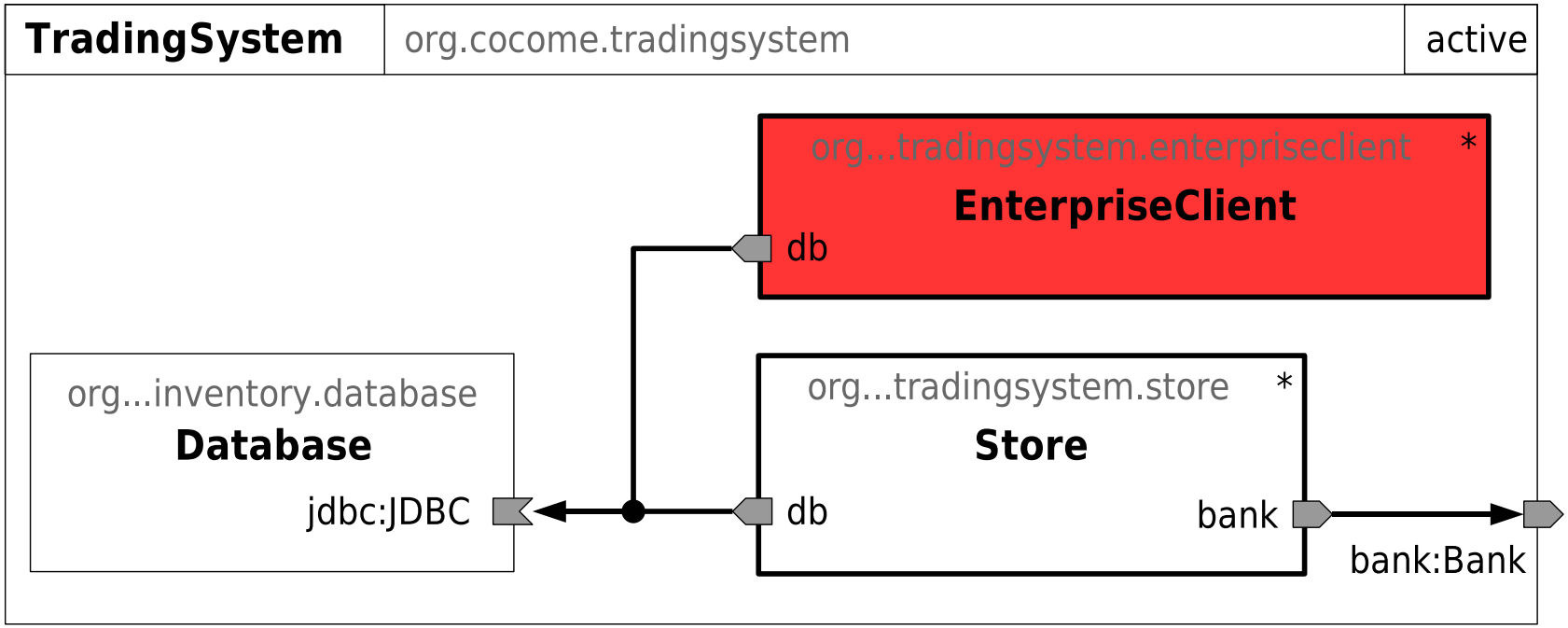


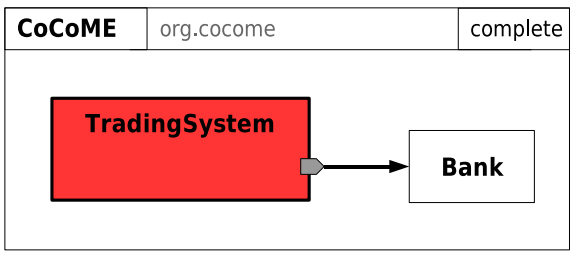
The TradingSystem Component



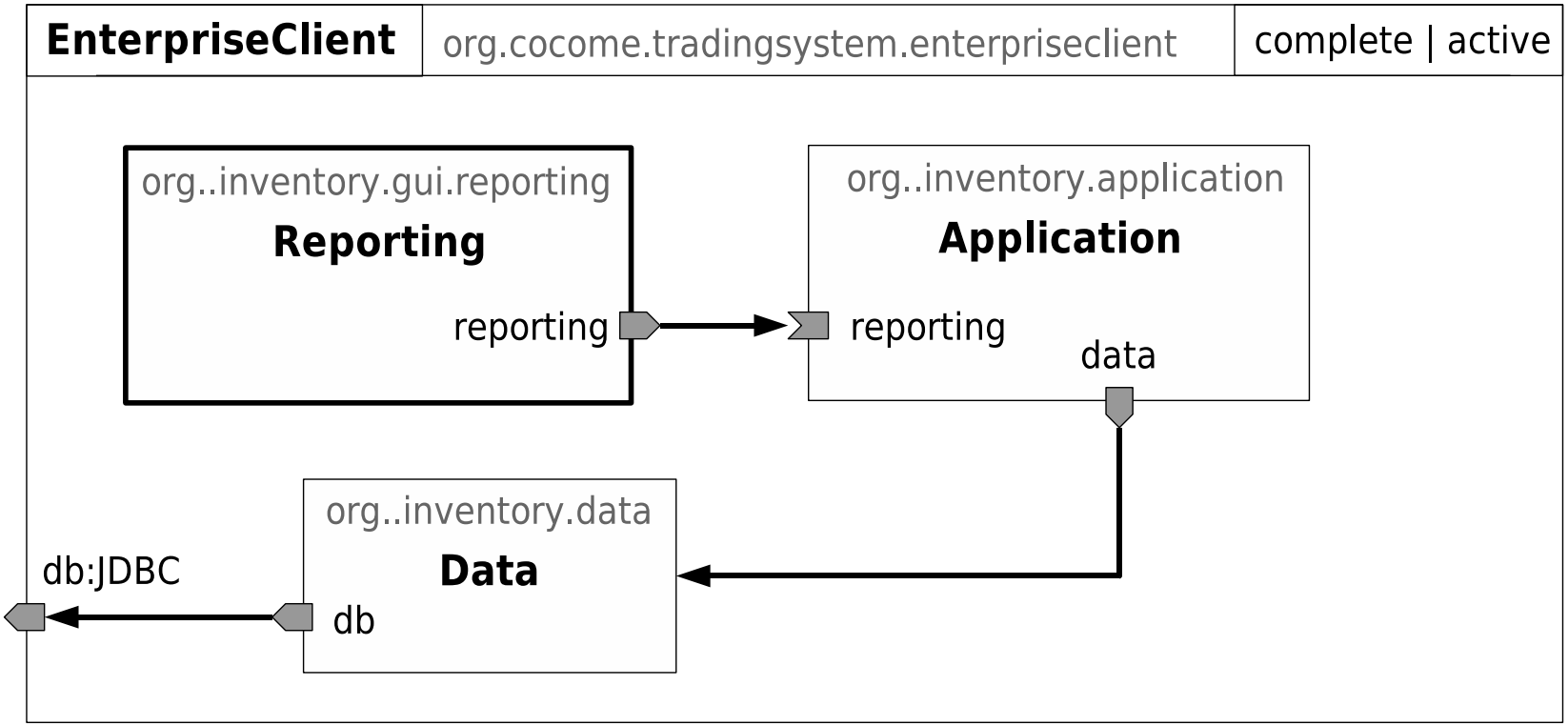


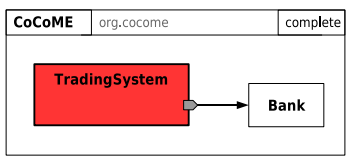
The TradingSystem Component

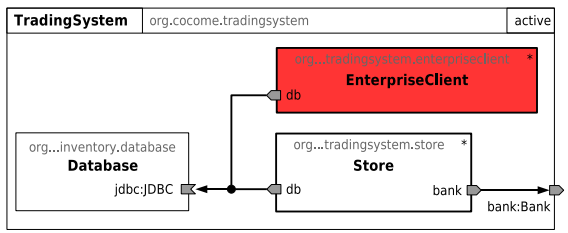




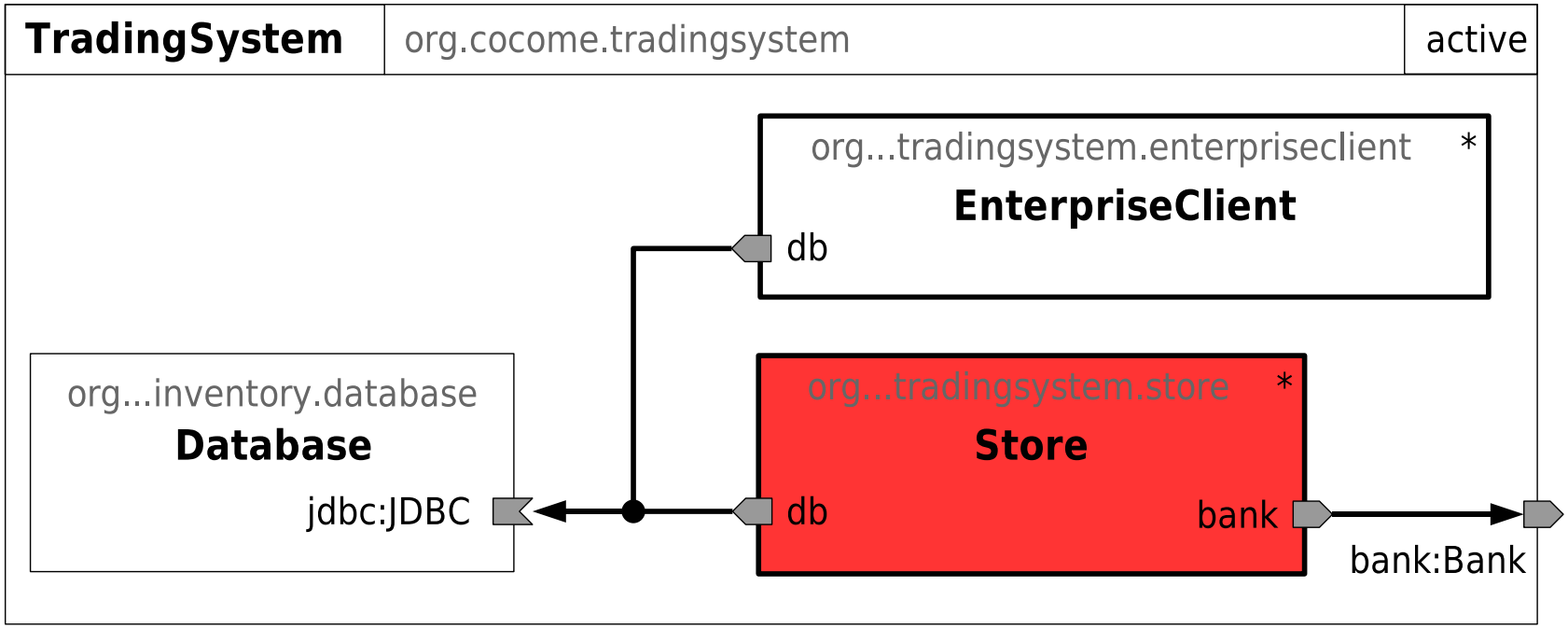
The EnterpriseClient Component

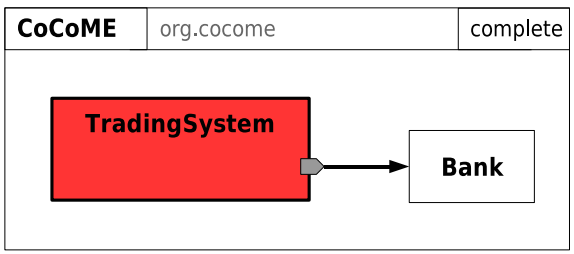




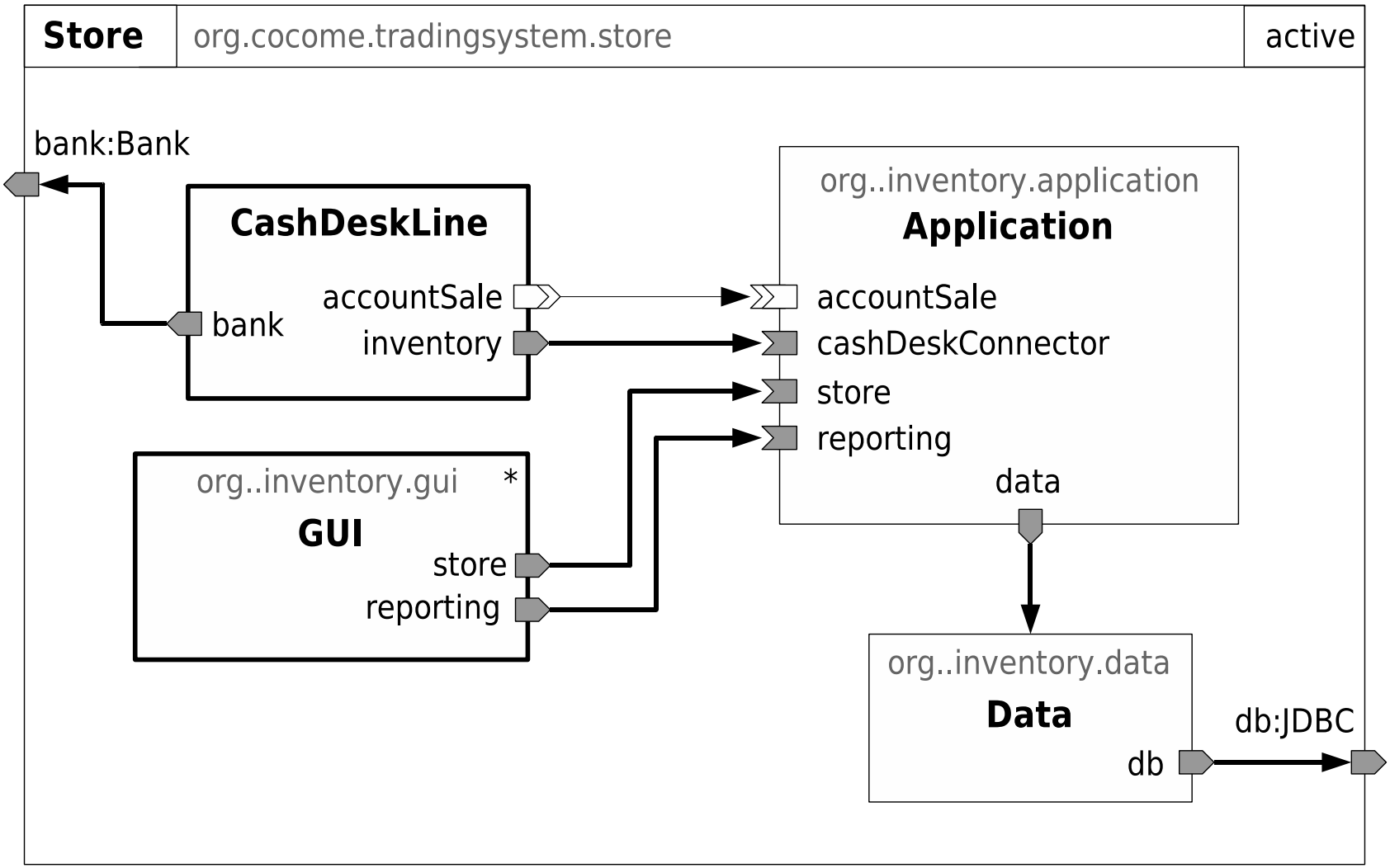


The Store Component

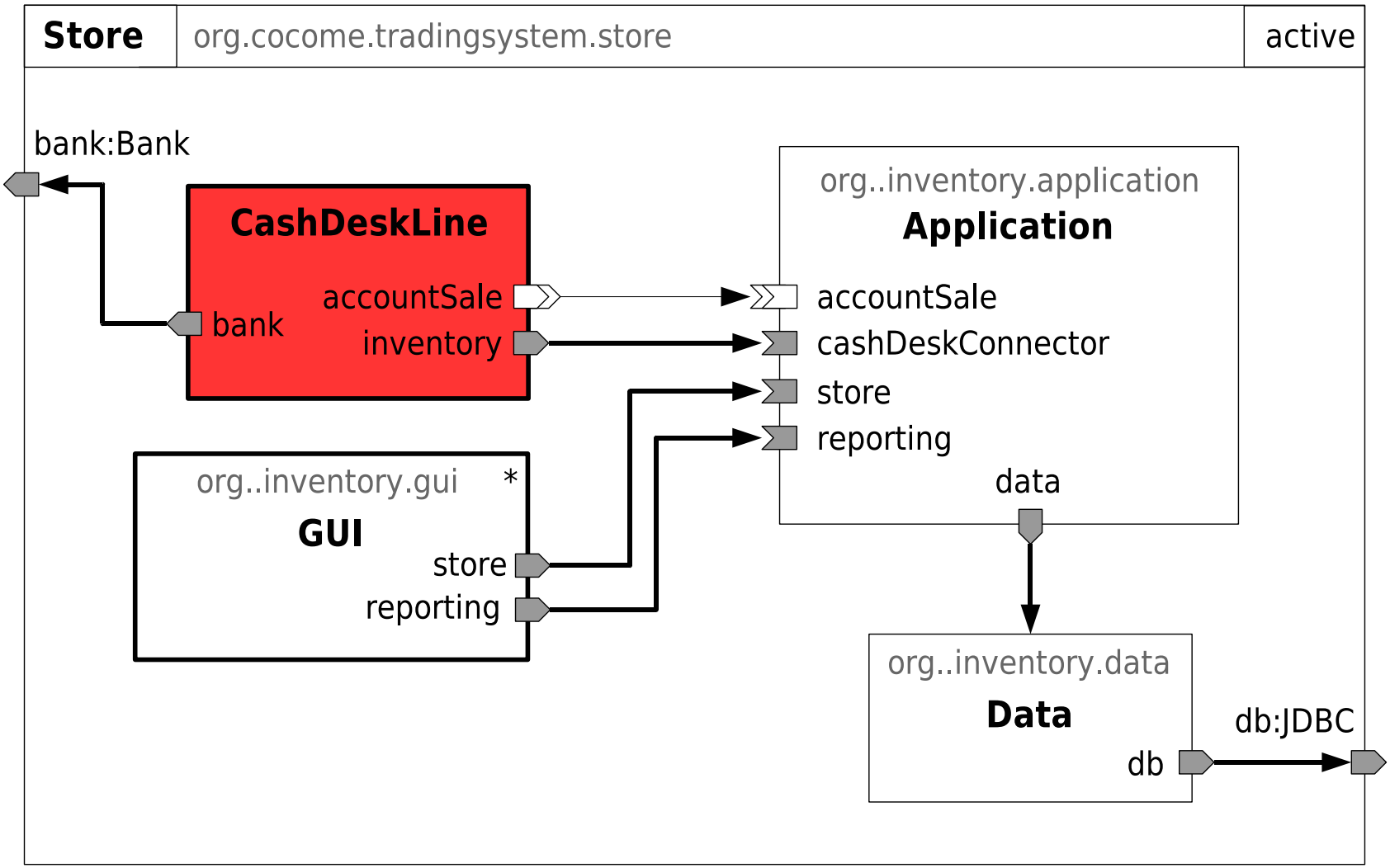




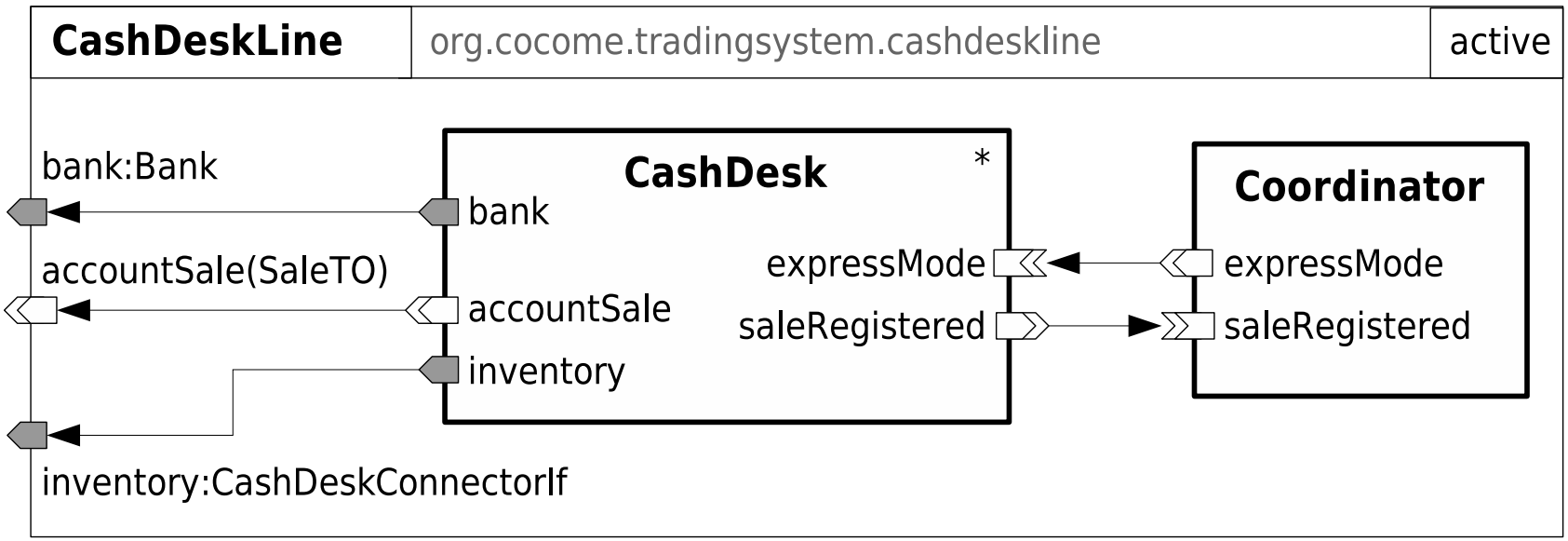
The Store Component

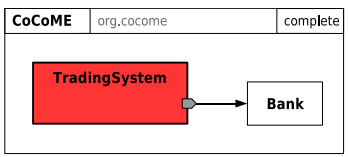


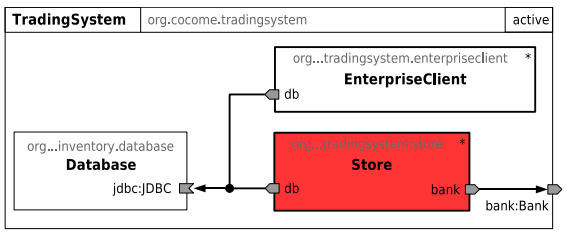
The CashDeskLine Component

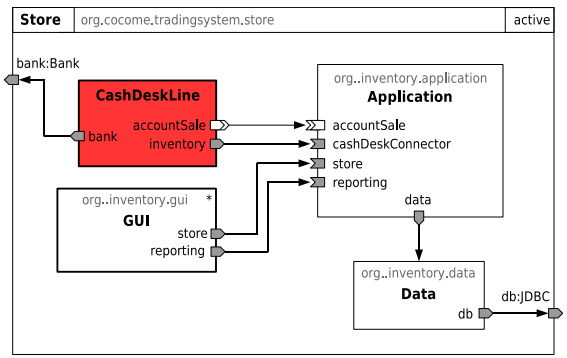


The CashDeskLine Component

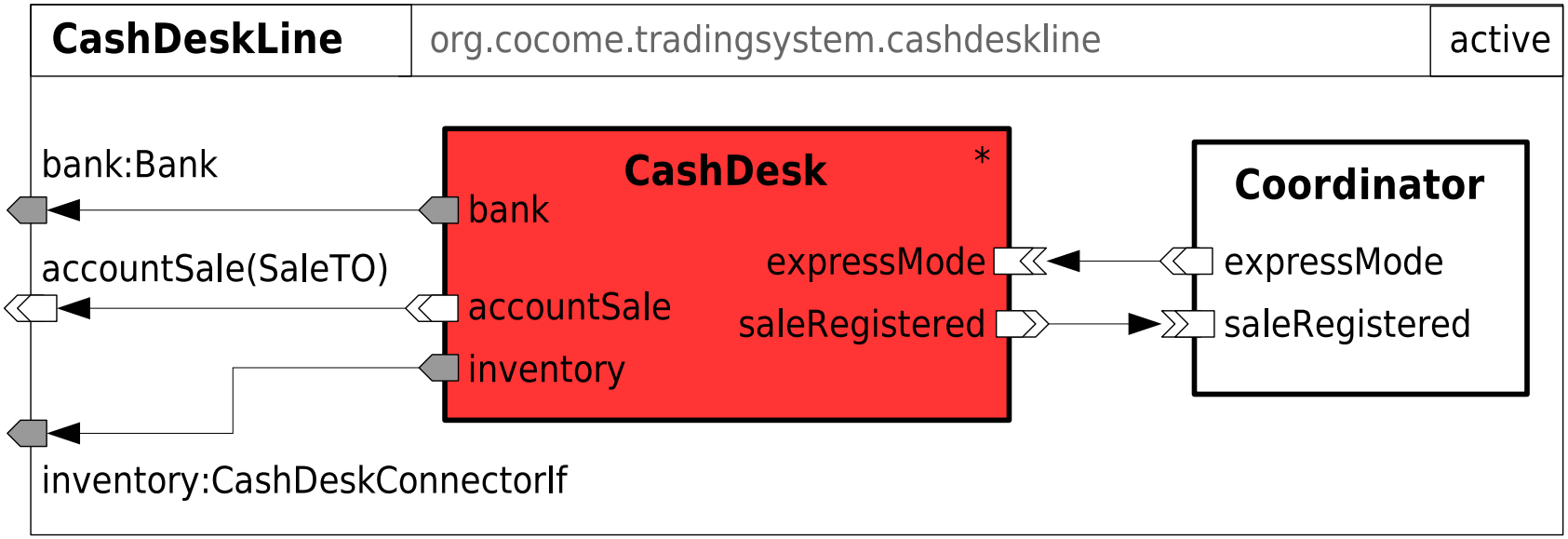


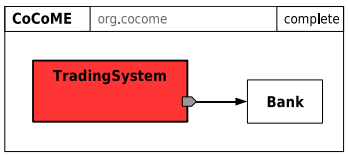


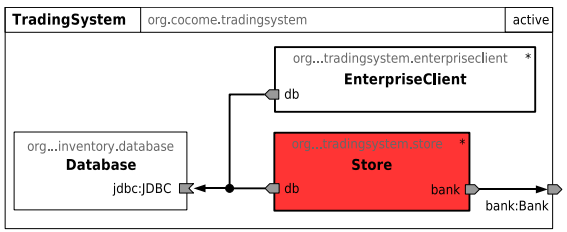


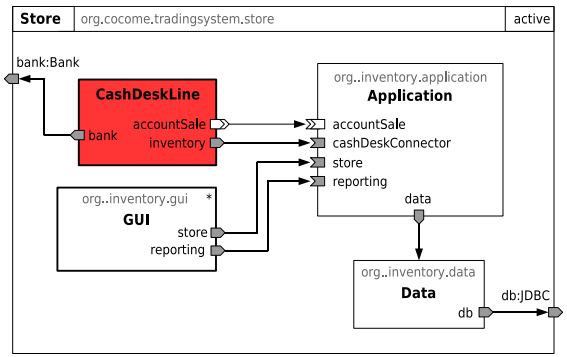


The CashDesk Component

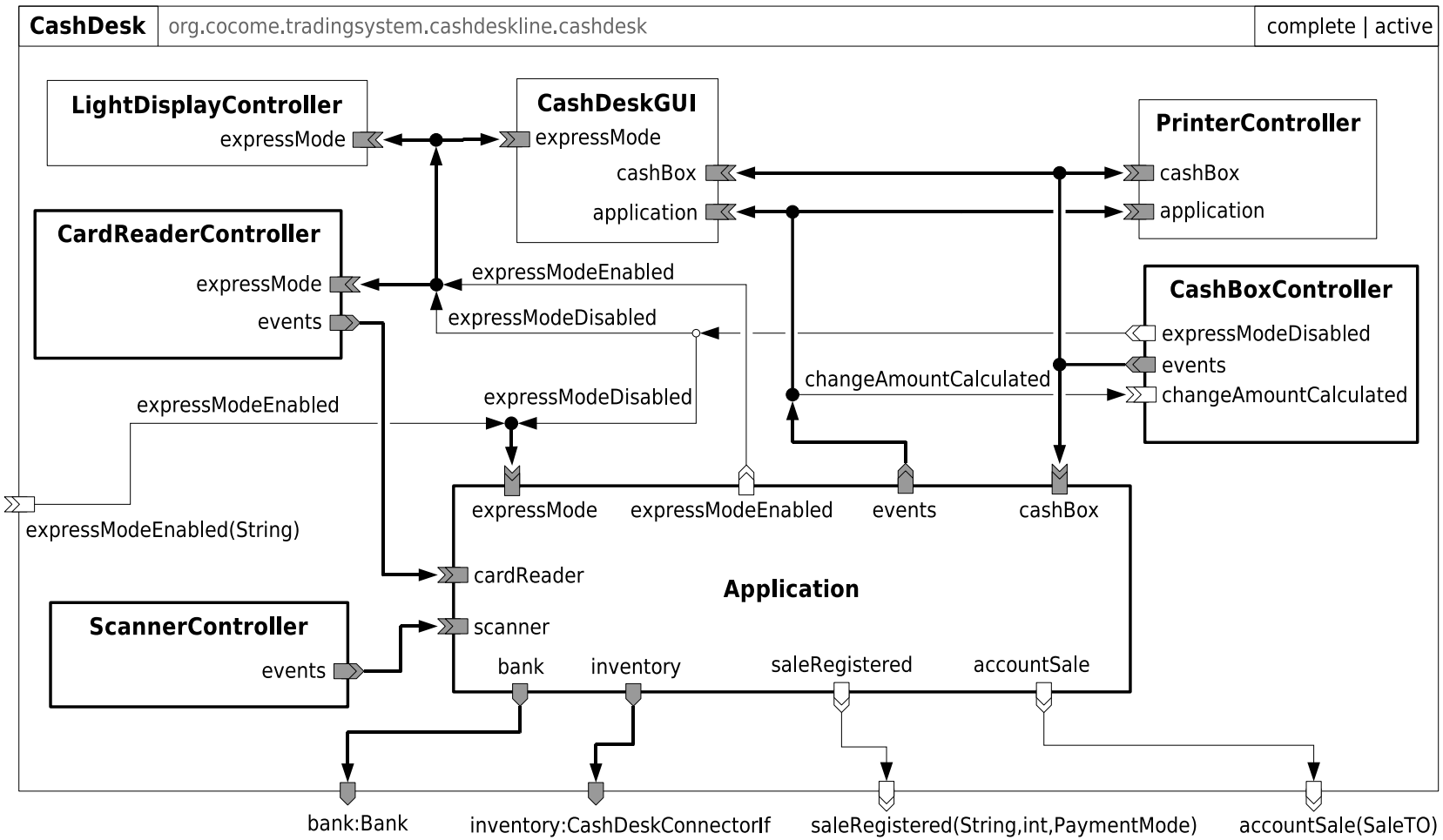








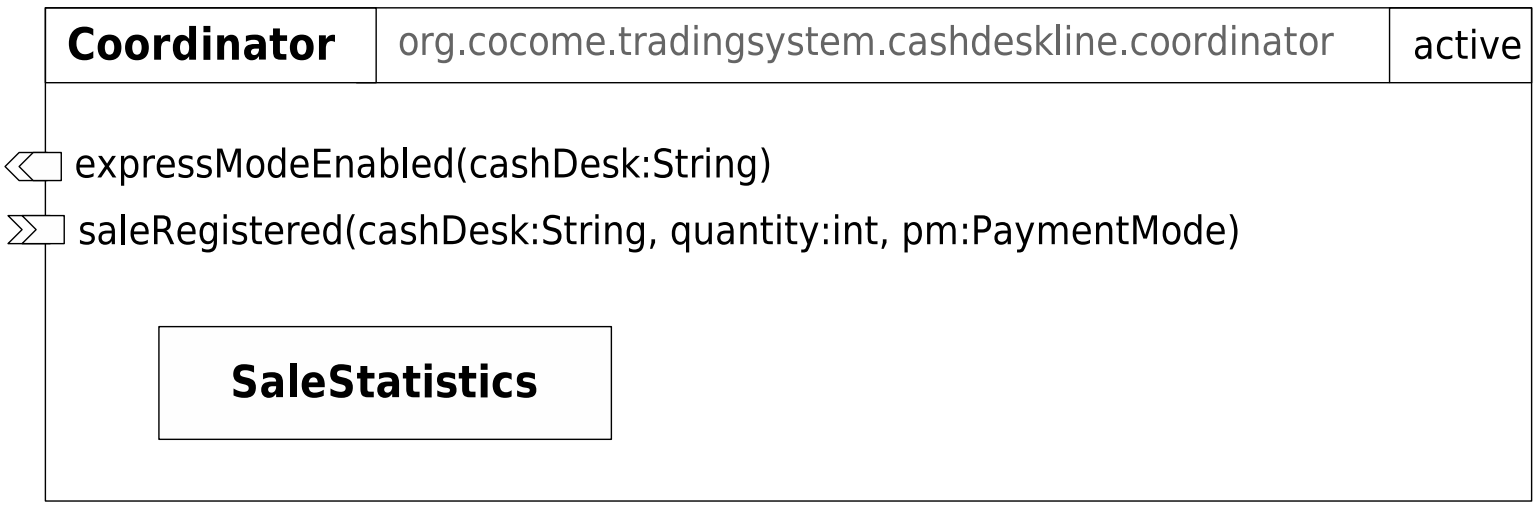
The CashDesk Component

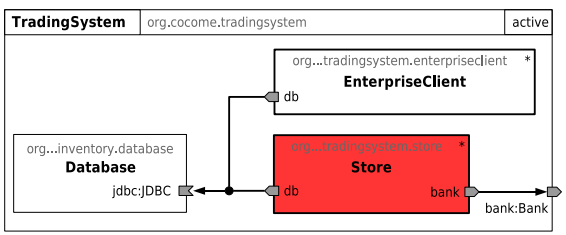


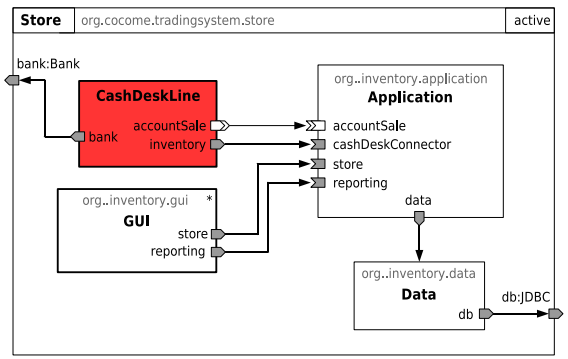
Executable Model

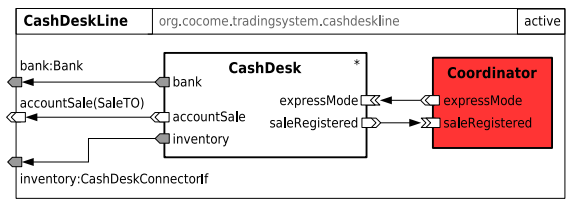
The Coordinator Component

MCowch Diagram









The Coordinator Component

JCowch Model

```
public box class Coordinator {
  public out async expressModeEnabled(String cashDesk);
  public async saleRegistered(String cashDesk, int quantity, PaymentMode pm);

  private SaleStatistics saleStats;

  public Coordinator() { saleStats = new SaleStatistics(); loop(); }

  private void handleNext()
    & saleRegistered(String cashDesk, int quantity, PaymentMode pm)
  {
    saleStats.registerSale(quantity,pm);
    if (saleStats.isExpressModeNeeded())
      expressModeEnabled(cashDesk);
  }

  private async loop() { while (true) handleNext(); }
}
```

Conclusion

Conclusions

- Almost complete arch. model of the CoCoME system
- Architectural descriptions provide a concise overview
- Consistency relation to executable model
- Concurrency support should be refined

Present and Future Work

Short-Term (This Year)

- Formal result to substitutability of OO-components
- Formal semantics for JCowch
- Concurrency model improvements
  - Apply ideas from Creol (cooperative multi-tasking)
  - Integrate Futures
- Tools
  - JCowch compiler
  - MCowch designer
  - Rudimentary automatic consistency checking

Future Work

Long-Term (Next Year(s))

- Full (semi-)automatic architectural consistency checking
- Integrated Development Environment for Cowch
- Behavioral specification and verification

Thank You!