INRIA

A Specification Language for Distributed Components

Antonio Cansado
Denis Caromel
Ludovic Henrio
Eric Madelaine
Marcela Rivera
Emil Salageanu

OASIS Team
INRIA Sophia Antipolis, France

- Parallel, Distributed Applications
  - ASP Calculus & ProActive middleware
    - Active Objects
    - Grid Component Model (GCM)
- Behavioural Models on ProActive
  - Networks of Labelled Transition Systems
  - Vercors Platform
    - Model Generation & Model-Checking

Focus of the Modelling Approach

- Behaviour specification
  - JDC – Specification Language
  - UML diagrams
- Grids: Asynchronous, distributed and parallel
- Verification of Temporal Properties
- Generation of Safe Code

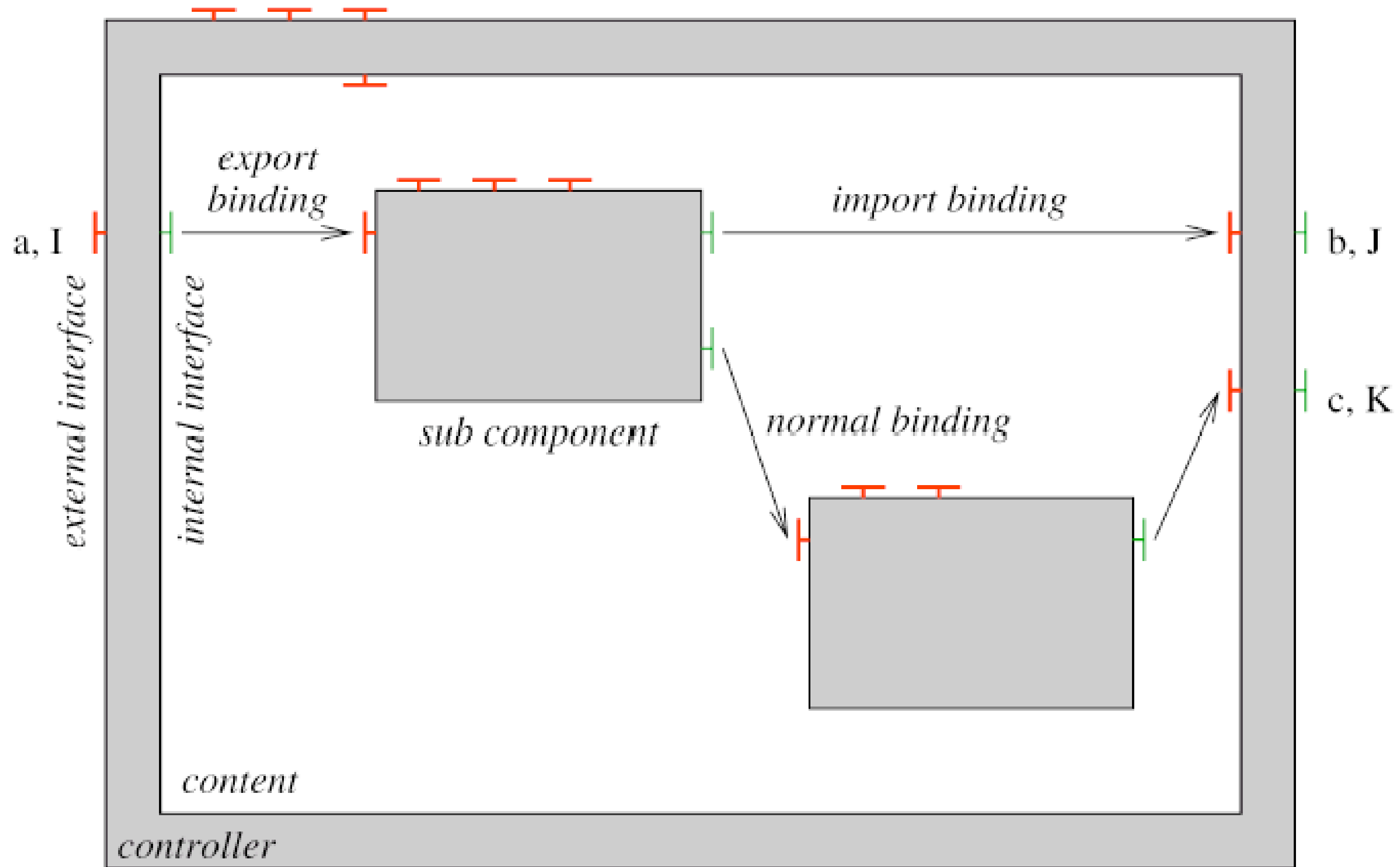
Plan

- Component Model
- Specification Languages
- Formal Model
- Verification
- Code Generation

Grid Component Model (GCM)

- From Fractal:
  - Hierarchical structure
  - Separation of concerns
    - Functional
    - Non-Functional: LF, BC, CC,...
  - Abstract component model
    - no constrain on implementation: several implementations exist
  - Architectural Description Language (ADL)
  - Hook to Behavioural Specification

A Fractal Component



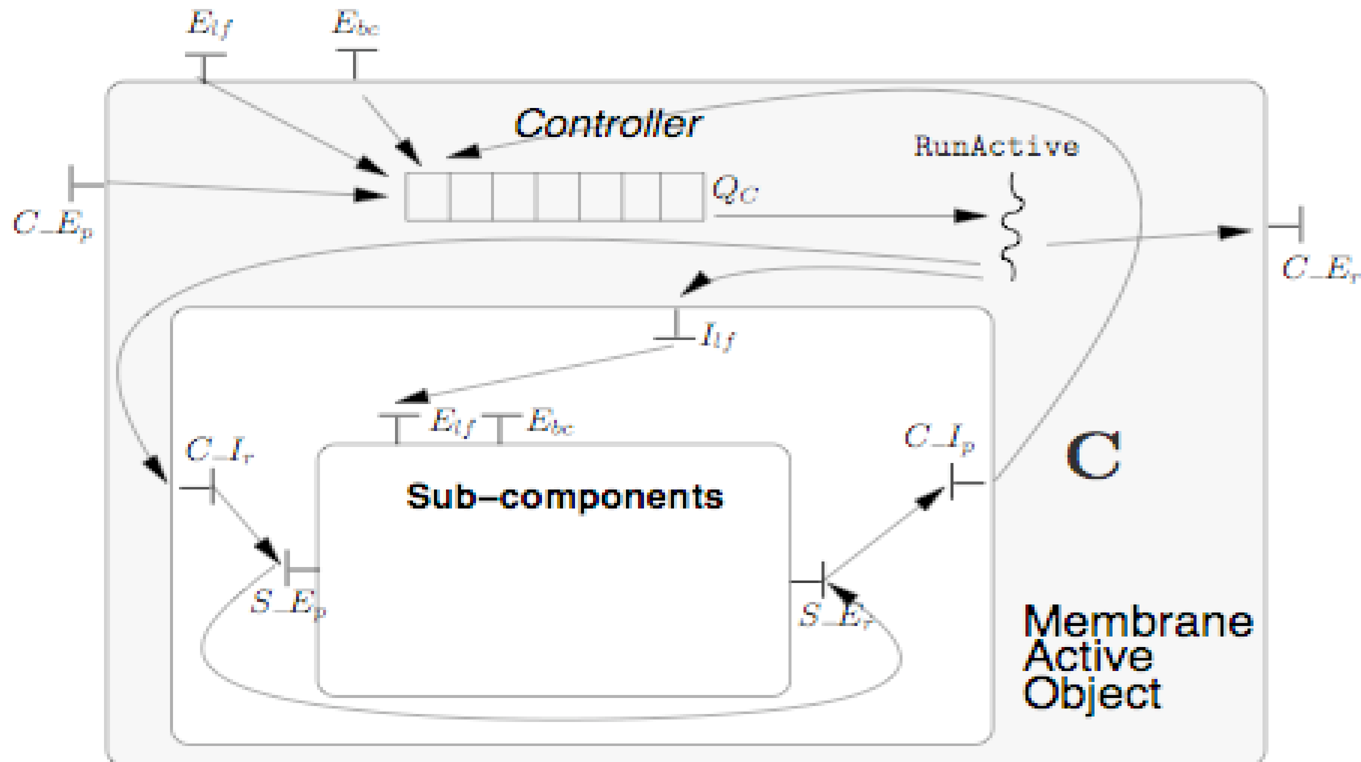
GCM Extensions

- Communication
  - Asynchronous method call
    - Synchronous method calls are also possible
  - Collective interfaces
- Other
  - Multiple (parallel) components
  - Virtual nodes & Deployment

GCM / ProActive

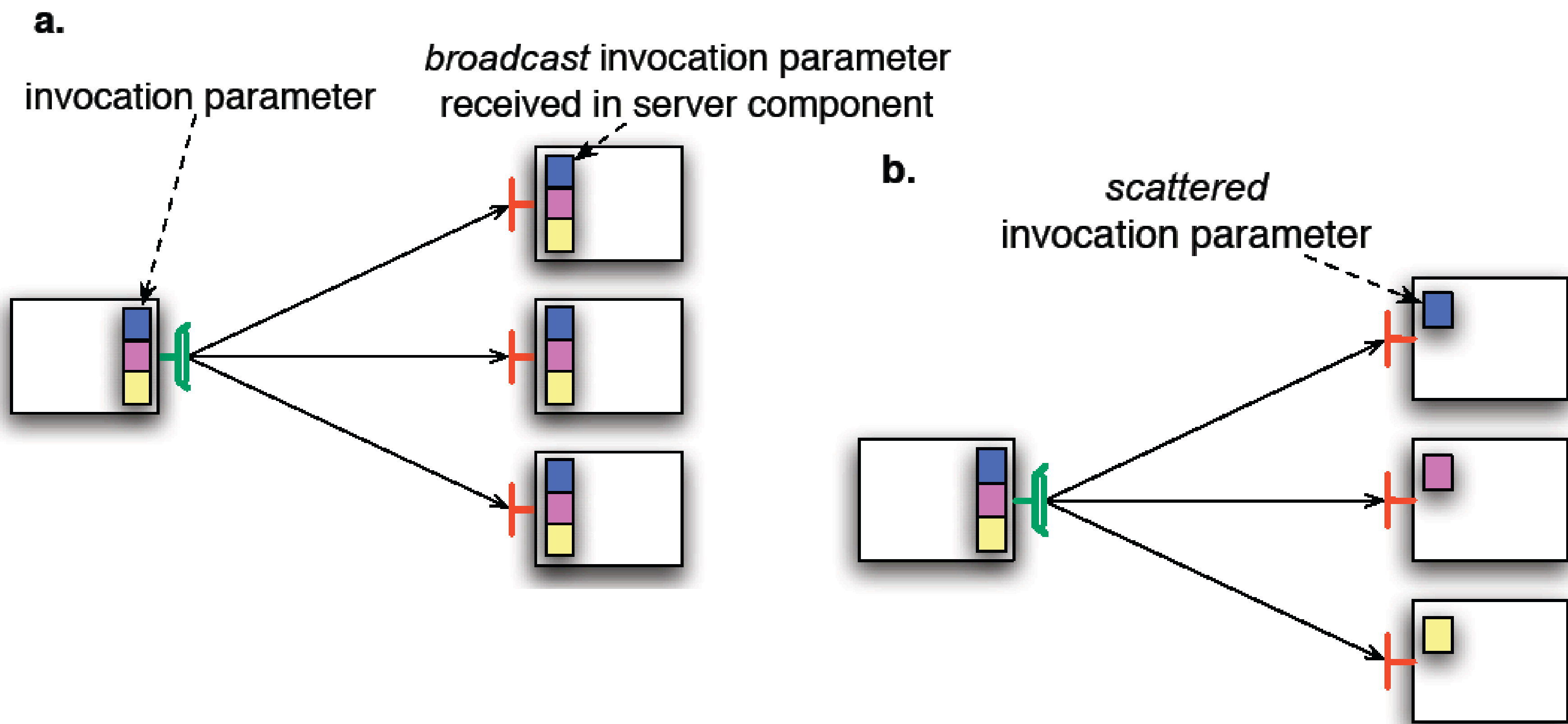
- Some features of the implementation
  - Active components
  - No sharing between components
  - Transparent futures
  - Fault-tolerance
  - Load-balance
  - Deployment – mapping of resources
    - Globus, Unicore, ssh, gLite, ...

GCM / ProActive



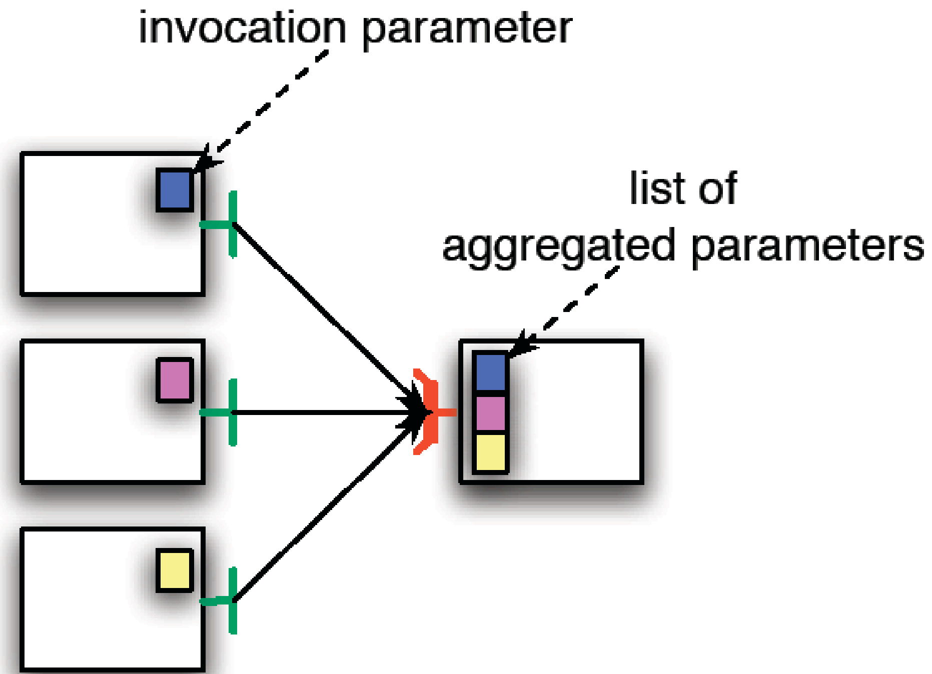
Multicast Interfaces

- Transform a single invocation into a list of invocations



Gathercast Interfaces

- Transform a list of invocations into a single invocation



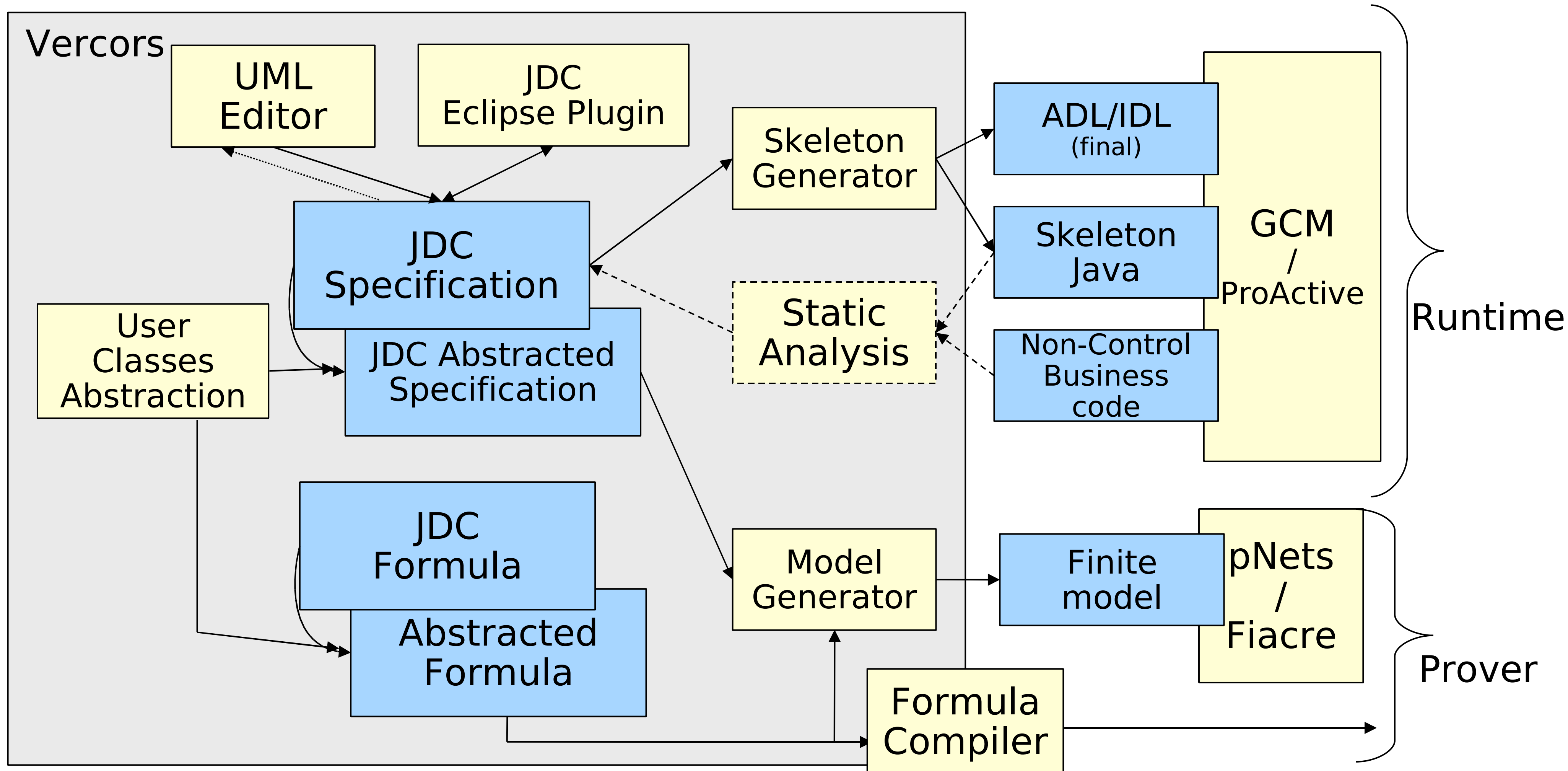
Behavioural Models

- Need:
  - Safe Assembly of Components
    - ✗ Static typing of bound interfaces
    - ✔ Dynamic behavioural compatibility
- Choice:
  - Networks of Labelled Transition Systems
  - Integrate ADL and BDL
- Challenge:
  - Provide a framework for non-specialists

Plan

- Component Model
- **Specification Languages**
  - **Java Distributed Components (JDC)**
  - **UML Diagrams**
    - **Black-Box view**
    - **Architecture view**
    - **Behaviour view**
- Formal Model
- Code Generation
- Verification

Vercors platform



JDC = Java Distributed Components

- Textual language, Java-based
- Includes **architecture** and **behaviour** primitives
- Standard Java syntax, excluding concurrency features
- Data definition using standard Java user-defined classes
- Fundamental GCM/ProActive concepts are explicit:
  - Component architecture, interfaces, NF interfaces
  - Service policies
  - Collective interfaces and multi/gather-cast policies
  - First-class futures

JDC Specification Language

- Black-Box View
  - Externally visible architecture and behaviour
  - List of interfaces (client and server interfaces)
  - Definition of interfaces (Method signatures)
- Architectural View
  - One-level refinement of a component
  - Composition of subcomponents
  - Internally visible Bindings

JDC Specification Language

```
component <NAME> (<Params>) {
   interfaces
      <interface> *
   services
      <service> *
   architecture
     contents
        <component> *
     bindings
      <binding> *
endComponent
```

```
<service> =
   service "{"
     <Local Variable Declaration> *
     policy "{" <RegularExpression> "}" *
     <Service Method Declaration> *
     <Local Method Declaration> *
     "}"
```

Architectural View

```
component CashDesk (int numOfBanks) {
  interfaces
    server interface CardReaderControlIf cardReaderControlIf;
    client interface BankIf banksIf [numOfBanks];
    ...
  architecture
    contents
      component CashDeskApplication (numOfBanks) application;
      component CashReaderController cashReader;
      ...
    bindings
      bind(application.eventBusIf, this.eventBusIf);
      for (int i: numOfBanks) {
          bind(application.banksIf[i], this.banksIf[i]);}
      ...
```

- Enough information to generate the ADL, plus data-flow used for synchronising components

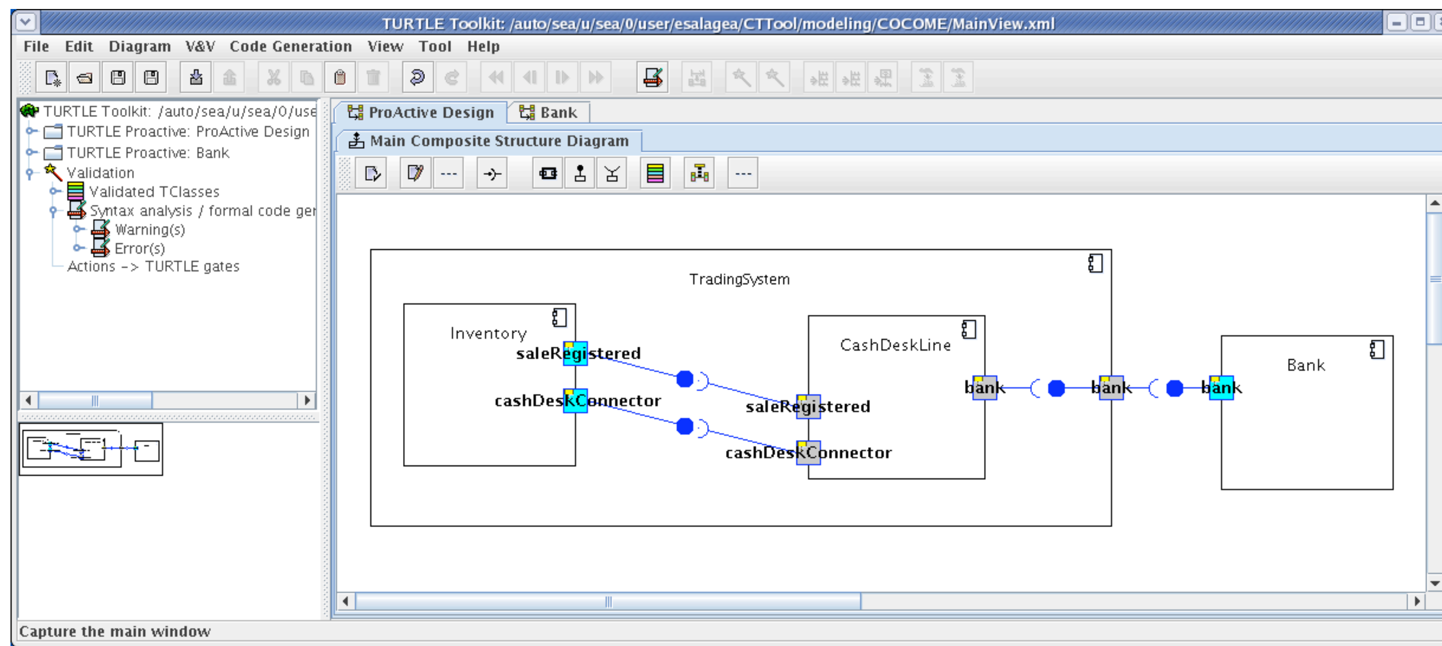
Behaviour Modelling

```
service { // CardReaderController
    policy {
      (
        (emit() | serveOldest(cardReaderControlIf.expressModeDisabled))*;
        serveOldest(cardReaderControlIf.expressModeEnabled);
        serveOldest(cardReaderControlIf.expressModeDisabled)
      )*
    }
    void emit() {
      if (__ANY(bool)) // non-deterministic choice
      cardReaderEventIf.creditCardScanned(__ANY(CreditCard));
}
```

- Enough information to generate the component service policy method (***runActivity***);
- and a skeleton for each service method
- Data + control flow information used to generate the abstract behavioural models.

Graphical (UML) Version: CTTool

- Targeted at non-specialist developers
- Implements only a subset of JDC
- UML 2: Component & Statemachine diagrams



Black-Box View
Interface Definition

BankIf

+ validateCardReq : void;
+ validateCardResp : void;
+ debitCardReq : void;
+ debitCardResp : void;

ApplicationEventHandlerIf

+ saleStarted : void;
+ productBarcodeScanned : void;
+ saleFinished : void;
+ cashAmountEntered : void;
+ cashBoxClosed : void;
+ creditCardPaymentEnabled : void;
+ creditCardScanned : void;
+ pinEntered : void;
+ expressModeEnabled : void;
+ expressModeDisabled : void;
+ paymentMode : void;

CardReaderEventIf

+ creditCardScanned : void;
+ pinEntered : void;

ScannerEventIf

+ productBarcodeScanned : void;

CashBoxEventIf

+ saleStarted : void;
+ saleFinished : void;
+ cashAmountEntered : void;
+ cashBoxClosed : void;
+ paymentMode : void;

CachDeskConnectorIf

+ getProductReq : void;
+ getProductResp : void;

EventBusIf

+ expressModeEnabled : void;
+ expressModeDisabled : void;
+ runningTotalChanged : void;
+ cashAmountEntered : void;
+ changeAmountCalculated : void;
+ saleStarted : void;
+ saleFinished : void;
+ invalidCreditCard : void;
+ creditCardScanFailed : void;
+ productBarcodeNotValid : void;

cashDeskConnector bank AppEventHandler cardReaderEvent scanner cashBoxEvent

CashDesk

eventBus cashDeskGUIControl

saleRegistered lightDisplayControl cardReaderControl printer cashBoxControl

cashDeskGUIIf

+ runningTotalChanged : void;
+ cashAmountEntered : void;
+ changeAmountCalculated : void;
+ saleStarted : void;
+ saleFinished : void;
+ invalidCreditCard : void;
+ creditCardScanFailed : void;
+ productBarcodeNotValid : void;
+ cashBoxClosed : void;
+ expressModeEnabled : void;
+ expressModeDisabled : void;

SaleRegisteredIf

+ getNumberOfItems : void;
+ getPaymentMode : void;
+ registerSale : void;

LightDisplayControlIf

+ expressModeEnabled : void;
+ expressModeDisabled : void;

cardReaderControlIf

+ expressModeEnabled : void;
+ expressModeDisabled : void;

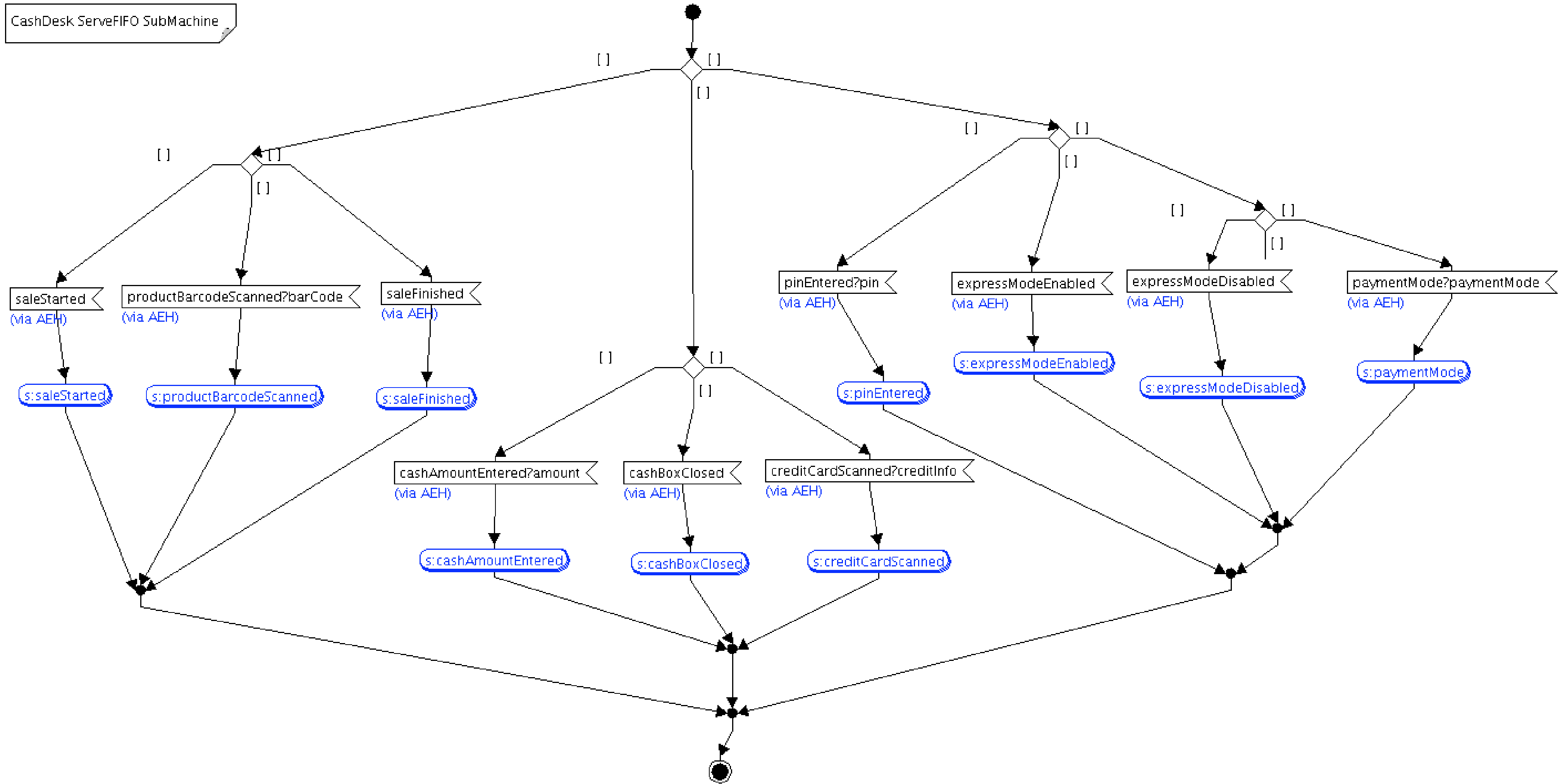
PrinterIf

+ runningTotalChanged : void;
+ cashAmountEntered : void;
+ changeAmountCalculated : void;
+ saleStarted : void;
+ saleFinished : void;
+ cashBoxClosed : void;

CashBoxControlIf

+ changeAmountCalculated : void;

Black-Box View
Behaviour Definition



Black-Box View
Behaviour Definition

paymentMode_INVALID = 0
paymentMode_CASH = 1
paymentMode_CARD = 2

cashState:

0 - IDLE
1 - PAYING
2 - STARTED

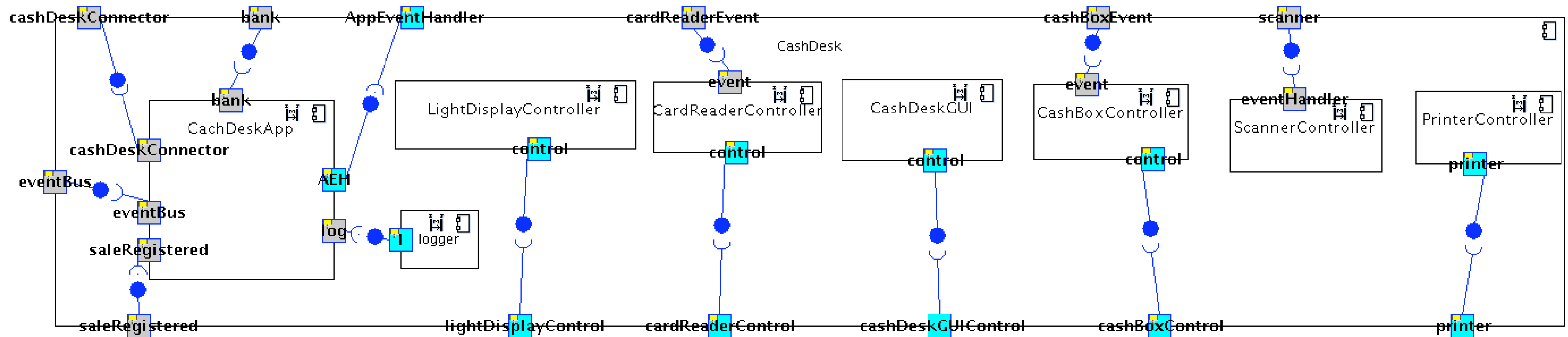
[cashState==1]
[cashState==0]
[cashState==2]
[ ]
[paymentMode==1]
[not(paymentMode==1)]

changeAmountCalculated!amount
(via eventBus)

errorCode=WrongPaymentModeException

errorCode=NotPayingException

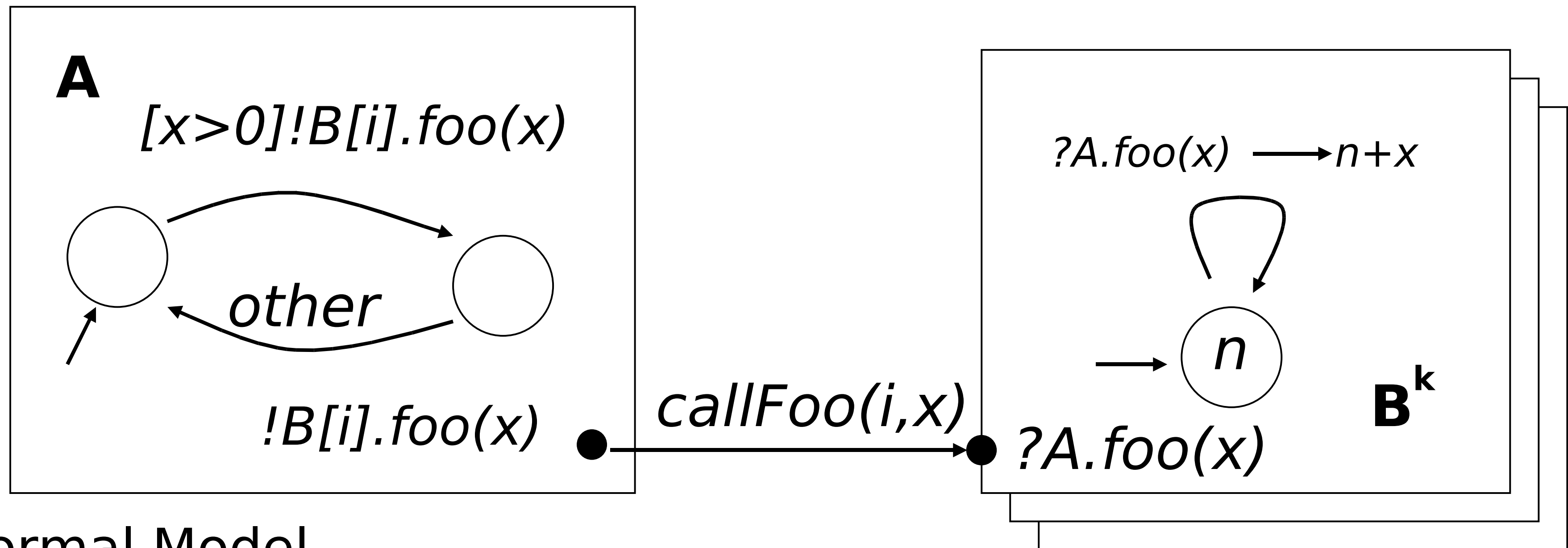
Architectural View



Plan

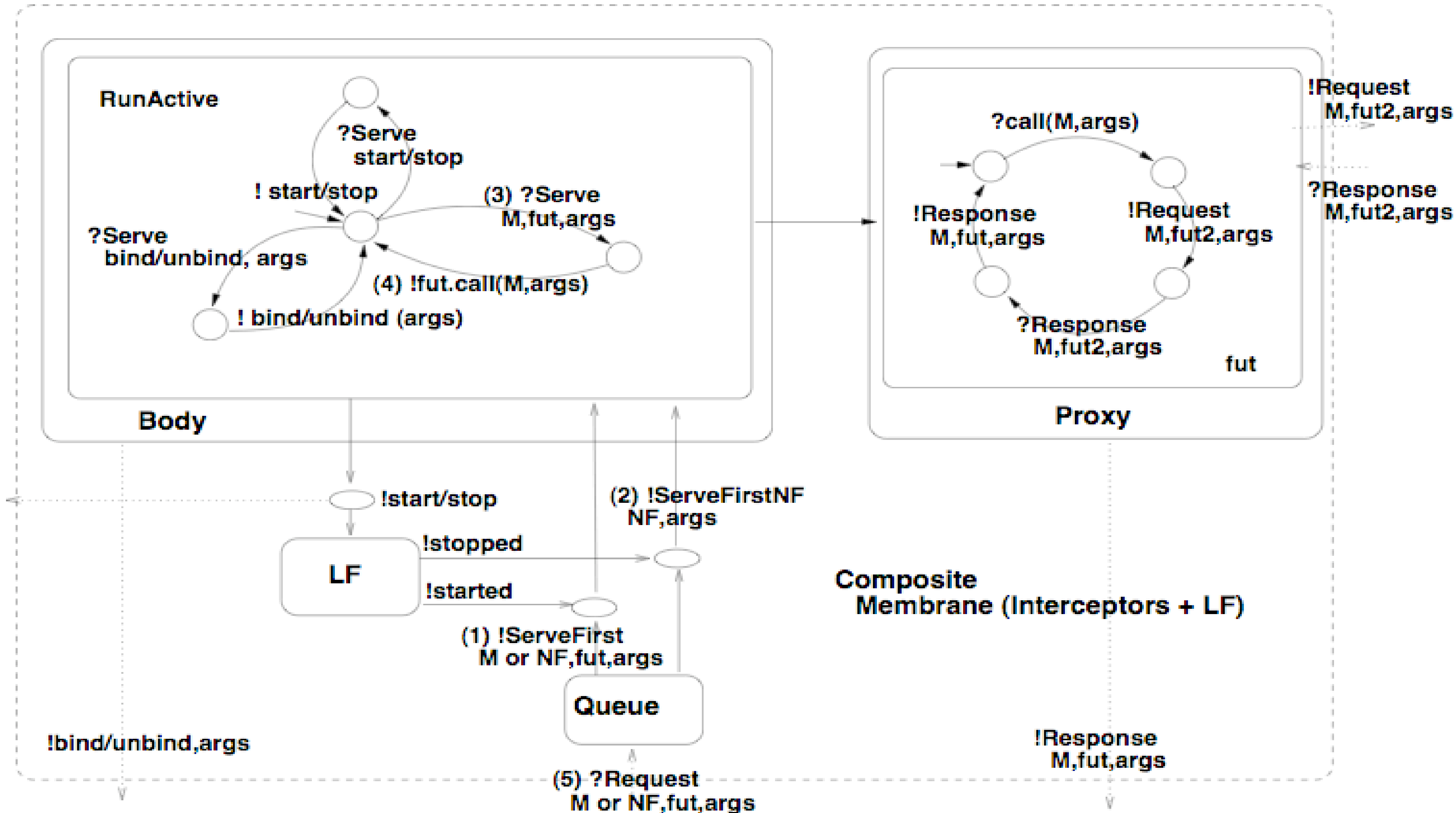
- Component Model
- Specification Languages
- **Formal Model**
    - **pNets: Parameterized Networks of Labelled Transition Systems**
- Code Generation
- Verification

Formal Model: pNets



- Formal Model
  - T. Barros, R. Boulifa, E. Madelaine: Parameterized Models for Distributed Java Objects, Forte'2004 Conference
- Distributed Components Behavioural Model
  - T. Barros, L. Henrio, E. Madelaine: Verification of distributed hierarchical components, FACS'05

Composite Membrane



Model Generation

(1) Data abstraction = mapping of user classes to simple data types

=> abstract JDC specification

(2) Architecture description of composite

+ Behaviour description of primitives

+ Synchronisation of subcomponents

+ Generation of NF Controllers

=> parameterized Network of synchronised LTSs

(3) Finite partition of parameter domains

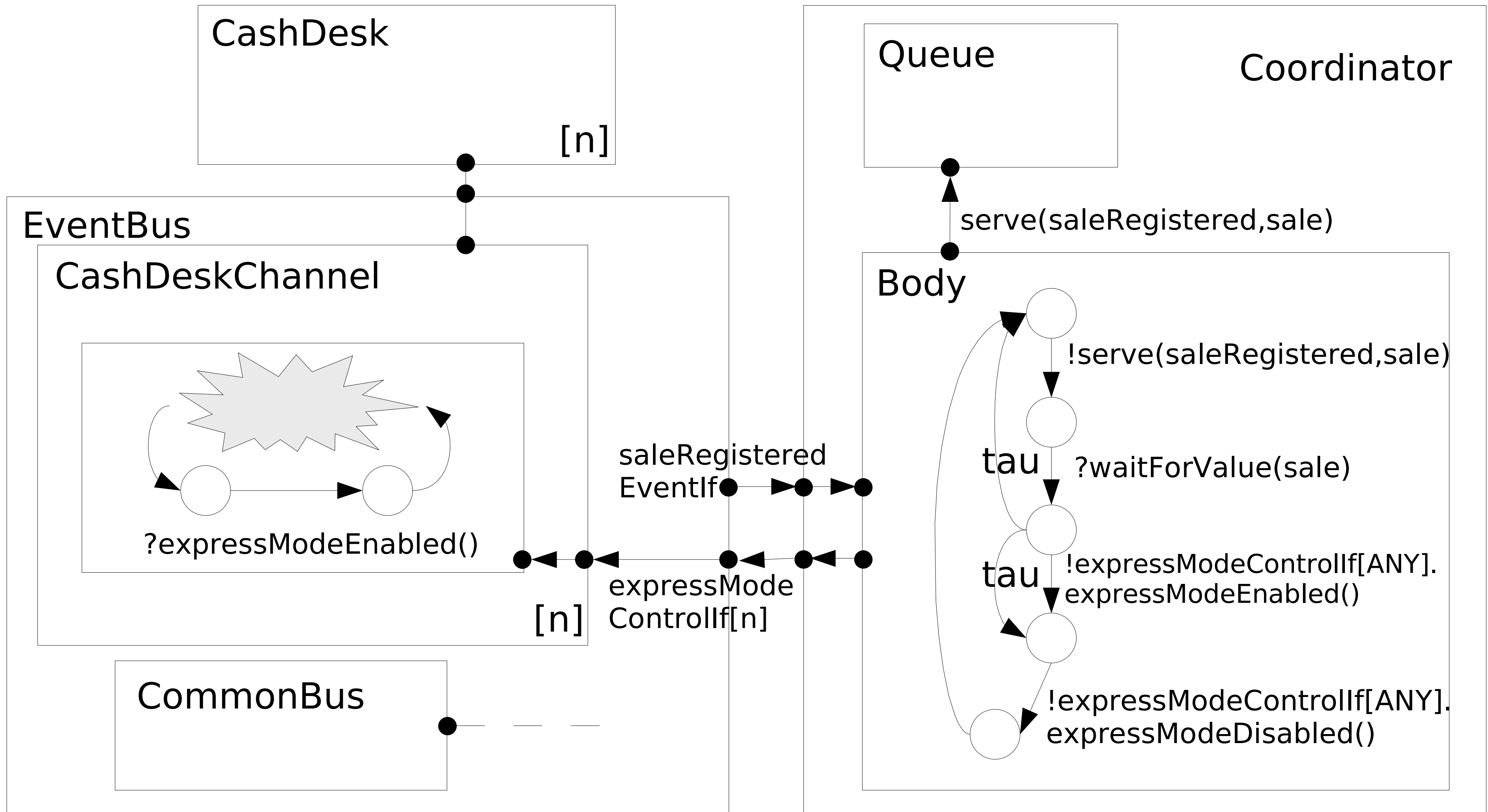
=> finite model suitable for explicit-state model checking

Data Abstraction

- Generation of an abstract JDC Specification
  - Mapping of User-classes into Simple Types

```
class <ClassName> "{"
  (public | private | protected)":" *
  <ClassType> <FieldName>";" *
  <ClassType> <FieldName> abstracted as <SimpleType>";" *
  <ReturnType> <MethodName> "(" <Params> ")" <Exception>";" *
  <ReturnType> <MethodName> "(" <Params> ")" <Exception> abstracted as "{"
    <UserCode> *
  "}" *
"}"
```

pNets within CoCoME



Plan

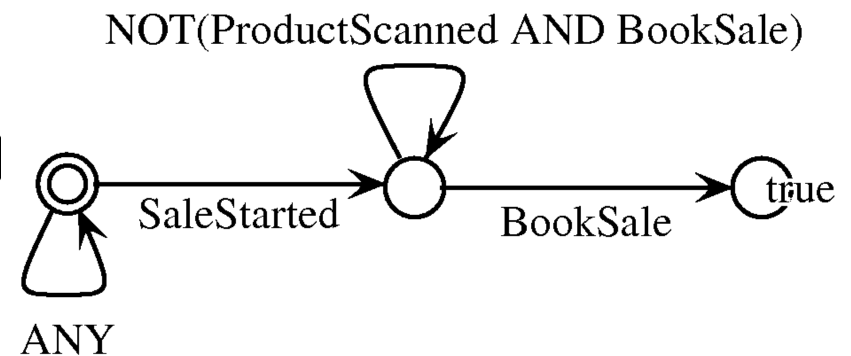
- Component Model
- Specification Languages
- Formal Model
- **Verification**
  - **Some properties of CoCoME**
- Code Generation

Model-Checking

- Press-button properties
  - Dead-locks, absence of predefined errors, etc.
- Safety properties
  - Specified as automata, or message sequences
- Conformance or Substitutability
  - Versus an equivalence / preorder relation

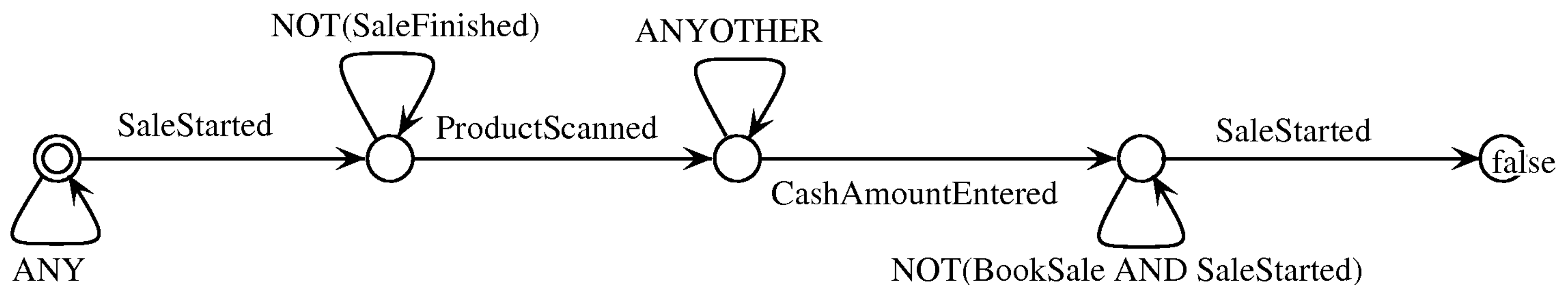
Safety Properties

- Formula in mu-calculus are too complex
- Automata-based
  - Transitions with predicates and logic quantifiers
  - Acceptance or rejection states
  - Special predicates
    - NOT(i), (i AND j), (i OR j)
    - ANYOTHER



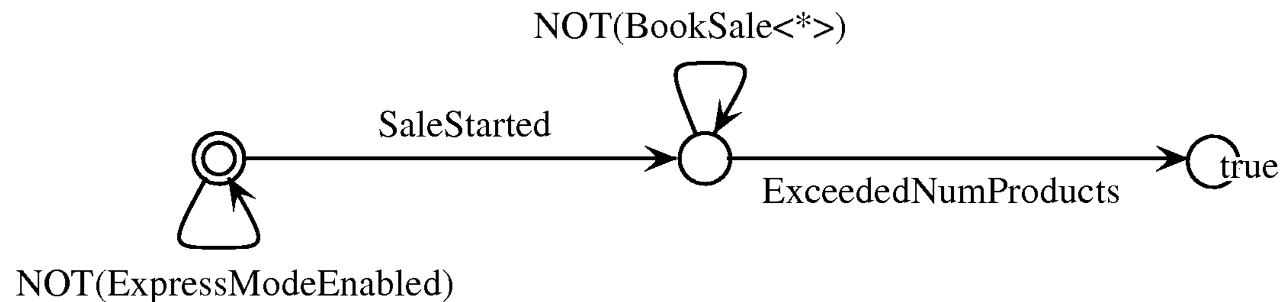
Verification Results

- Deadlocks were found
  - Due to synchronous versions of our encoding
- Main sale process is feasible (Use Case 1)



Verification Results

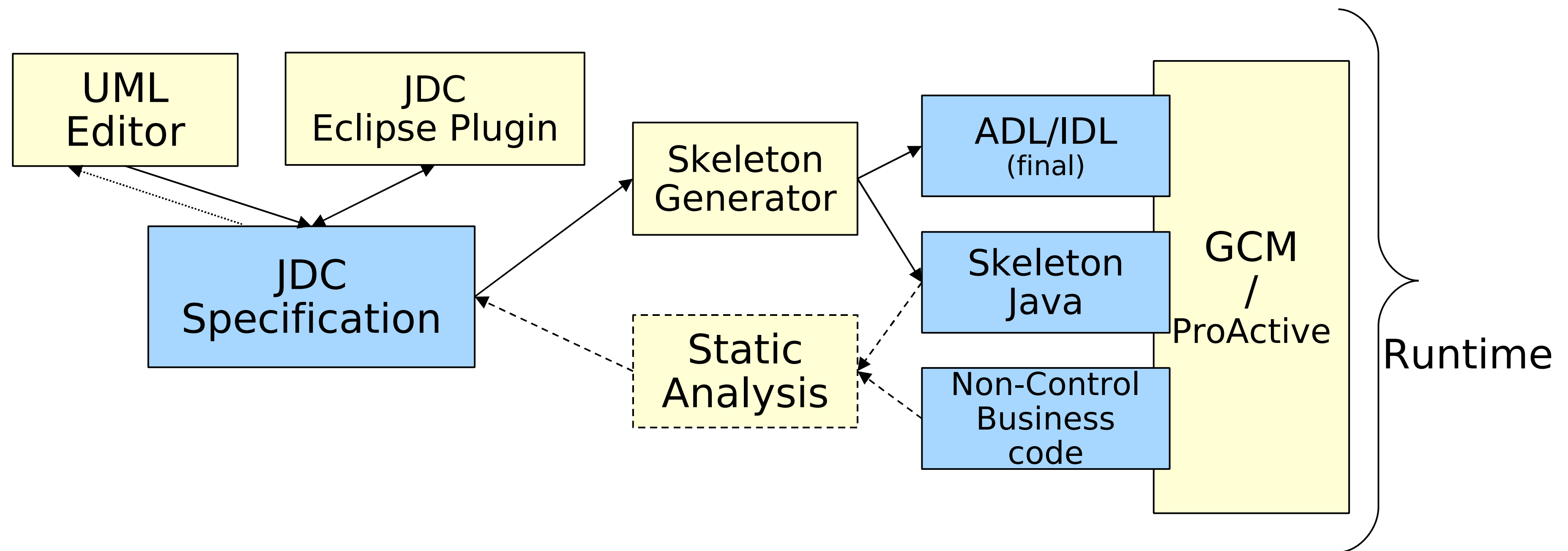
- Strange behaviours were found
  - Booking an Empty Sale
  - Successful Sale with Insufficient Money
- Error due to incomplete specification
  - Safety of the Express Mode (an express mode may be triggered during an ongoing sale)



Plan

- Component Model
- Specification Languages
- Formal Model
- Verification
- **Code Generation**
    - **Modelled part of CoCoME**
    - **Example of runActivity**

Code Generation



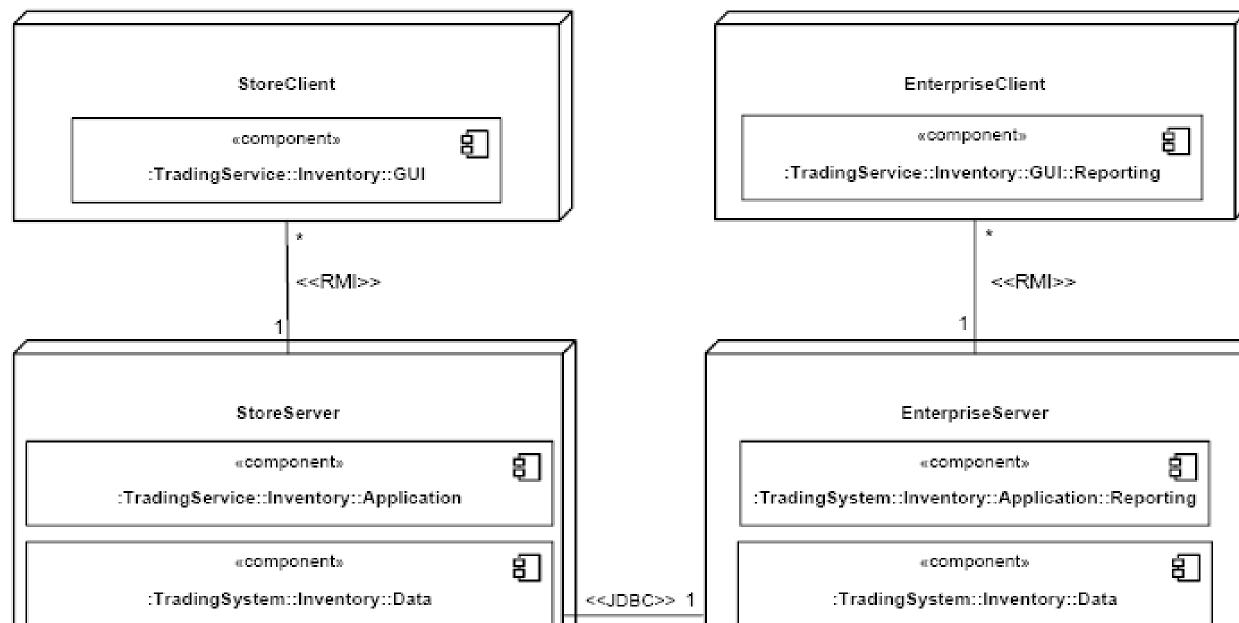
Code Generation

- ADL specification file (per component or per application)
- GCM/ProActive component class definition for each primitive component
  - Fully defined:
    - initActivity and runActivity methods (finalised)
  - Skeletons for:
    - service methods and local methods
    - (to be complemented by the developer, with strong constraints)

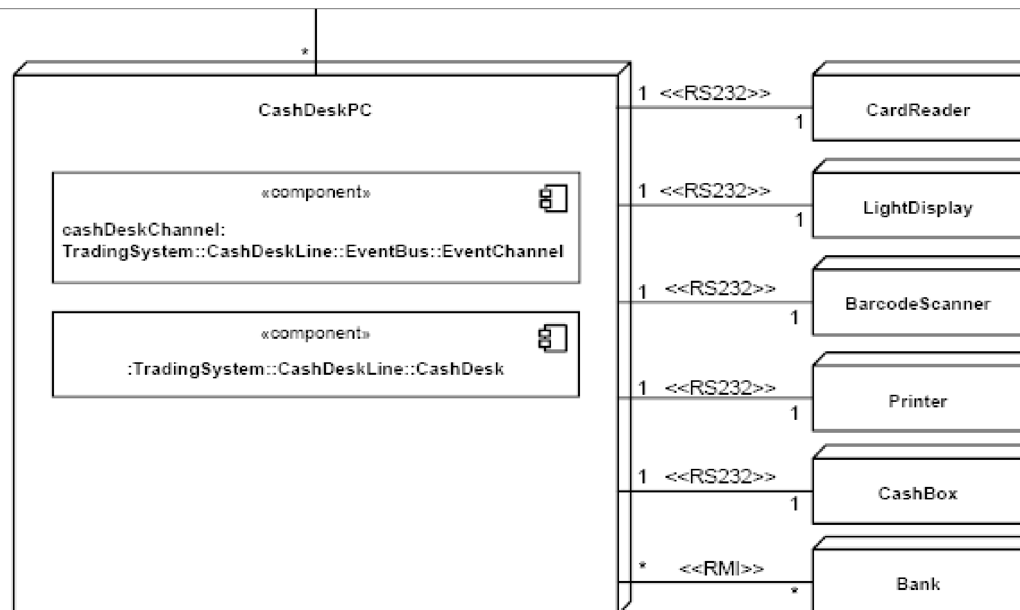
Example: Service Policy

- runActivity generated method

```
public void runActivity(Body body) {
   Service service = new Service(body);
      while (body.isActive()) {
         cashBoxEventIf.saleStarted();
         cashBoxEventIf.saleFinished();
         if ((new AnyBool()).prob(50)) {
            cashMode();
            cashAmount();
            service.blockingServeOldest(
                      "changeAmountCalculated");
            cashBoxEventIf.cashBoxClosed();
         } else
            creditCardMode();
      }
}
```



ProActive Deployment



Conclusions

- Contributions
  - Grid Component Model
  - Java Distributed Components specification language
  - Verification Platform
  - Proposal for Safe Code Generation

Conclusions

- Limitations
  - Only synchronous controllers are implemented
  - Partial GCM support
    - Collective interfaces, high-level reconfiguration

Conclusions

- Strengths
  - Integration of architecture with behaviour
  - Association of Component Model with Verification framework
  - (Partial) Operational Toolset
    - Able to scale up in middle size case-study
    - Found real bugs

Useful links

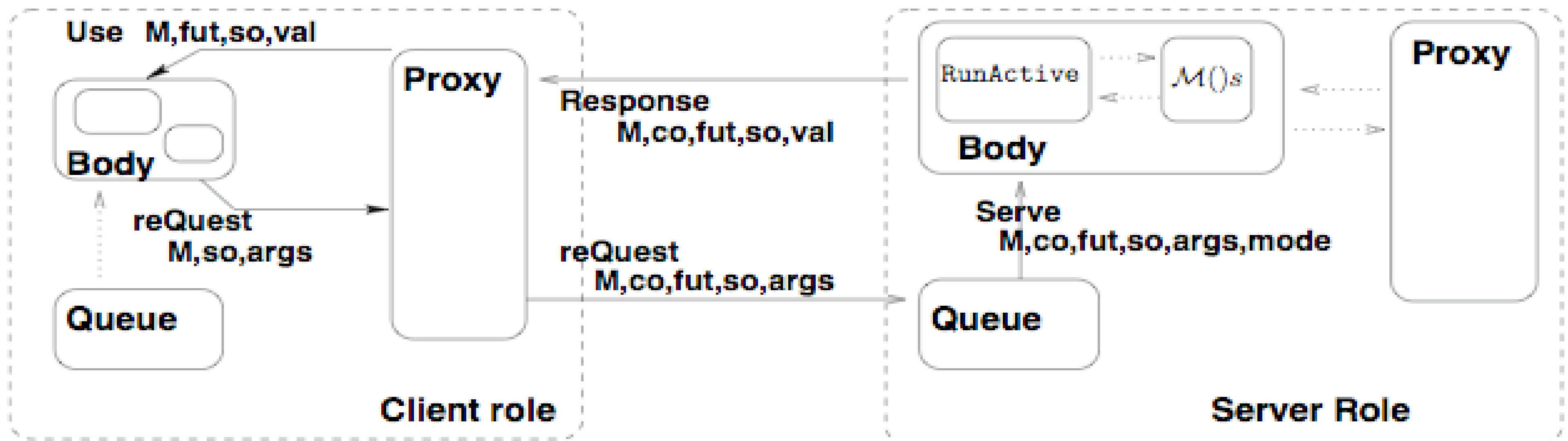
- ProActive (including GCM)
  - http://www-sop.inria.fr/oasis/ProActive/
- Vercors platform
  - http://www-sop.inria.fr/oasis/Vercors/
  - http://www-sop.inria.fr/oasis/Vercors/Cocome/

CTTool Diagram Constructs

Component Diagram	
Component	Producer
IN port	mb1
OUT port	p1
Connector	
Delegate connector	p1 p2
Interface signature	I3 Contingency : OPTIONAL + rGet : void;

State Machine Diagram	
(named) State	ready
Start/Stop points	
Submachine	s:treatFirstPut
Receive/Send message	qGet (via sq2) get (via cb1)
Action	stock=stock+1
Choice	[stock=0] [ ] [stock>0]
Non-determinism	[ ] [ ] [ ]

ProActive



GCM: Architecture Specification

- ADL + IDL

```
<component name="CashDesk">
  <interface signature="CashDeskLine.if.ApplicationEventHandlerIf"
role="server" name="applicationEventHandlerIf"/>
  ...
  <component name="CashDeskApplication">
    <interface signature="CashDeskLine.if.ApplicationEventHandlerIf"
role="server" name="applicationEventHandlerIf"/>
    <interface signature="if.CashDeskConnectorIf" role="client"
name="cashDeskConnectorIf"/>
    ...
    <content class="CashDeskLine.CashDesk.CashDeskApplication"/>
    <controller desc="primitive"/>
  </component>
  <component name="CardReaderController">      ...        </component>
  ...
  <binding client="this.appli
server="CashDeskApplication.a
  ...
```

```
public interface ApplicationEventHandlerIf {
    void expressModeDisabled();
    void expressModeEnabled();
    ...
}
```

GCM: Implementation

- Header

```
public class CashBoxController implements CashBoxControlIf,
BindingController, RunActive {
   public final static String CASHBOXEVENTIF_BINDING =
"cashBoxEventIf";
   private CashBoxEventIf cashBoxEventIf;

   // empty constructor required by ProActive
   public CashBoxController () { }
```

GCM: Implementation

- Controllers

```
    public String[] listFc () {
        return new String[]{ CASHBOXEVENTIF_BINDING };
    }

    public Object lookupFc (final String clientItfName) {
        if (CASHBOXEVENTIF_BINDING.equals(clientItfName))
            return cashBoxEventIf;
        return null;
    }

    public void bindFc (final String clientItfName,final Object
 serverItf){
        if (CASHBOXEVENTIF_BINDING.equals(clientItfName))
            cashBoxEventIf = (CashBoxEventIf)serverItf;
    }
    public void unbindFc (final String clientItfName){
        if (CASHBOXEVENTIF_BINDING.equals(clientItfName))
            cashBoxEventIf = null;
    }
```

GCM: Implementation

- Functional Interfaces

```
public void changeAmountCalculated(CashAmount changeAmount) {
    // the amount to be returned as change
    System.out.println(changeAmount.getAmount());
}
```

GCM: Implementation

- Local methods – Completed by the user

```
    private void cashMode() {
        cashBoxEventIf.paymentMode(
           new PaymentModeImpl(PaymentModeImpl.CASH));
    }
    private void creditCardMode() {
        cashBoxEventIf.paymentMode(
           new PaymentModeImpl(PaymentModeImpl.CREDIT));
    }
    private void cashAmount() {
        cashBoxEventIf.cashAmountEntered(
           new CashAmountImpl(1000)); // the client paid 1000
    }
} // end of CashBoxController implementation
```

Synchronisation issues in CoCoME

- Active components may trigger events
  - Clients, Scanners, CashDesks, ...
- Broadcast
  - Pricing updates
  - Flush caches on databases
- Multiple CashDesks querying Bank(s)