A Component Model for Architectural Programming

Hubert Baumeister, Florian Hacklinger, Rolf Hennicker,
Alexander Knapp, Martin Wirsing

Institut für Informatik
Ludwig-Maximilians-Universität München

March 2006

Our Department in March 2006

Software Architecture

Structuring elements

- **components** as building blocks
- **connectors** as glue
- **ports** for communication

Development process

- "[. . .] a component is modeled throughout the development life cycle [. . .]" (*UML 2.0 Superstructure Specification*)
- preservation and refinement of components

Software Architecture in Development Processes

Analysis and Design

Architectural Description Languages:
Wright, Darwin, Rapide, SARA, etc.

Implementation

- various component models (SOFA, EJB, ...)
- **but**: Programming languages do not reflect components
- danger of **architectural erosion**

$\Rightarrow$ integrate architectural concepts into programming languages: **Architectural Programming**

Architectural Programming (I)

Architectural Programming Languages (APLs)

- integrate **components**, **ports**, **connectors** and **configurations** as *primitive language constructs*

APLs should support

- **strong encapsulation** (communication exclusively via ports)
- ports with **provided**, **required interfaces** and **protocols**
- **connectors** for building up **configurations**
- runtime **reconfiguration**

Architectural Programming (II)

Advantages of Architectural Programming

- **maintainability**
- **reusability**, **replaceability** and **independent deployment**
- **seamless transition** from model to code
- **separation** of application code and glue code

Architectural Programming (III)

Our approach

Architectural programming language **JAVA/A**

Implementable Component Models

- **SOFA**, **Fractal**:
  - support for software architectures
  - not programming languages: architectural erosion
- **EJB**, **JavaBeans**, **COM+**, etc.
  - no support for software architectures
  - component models without ports, etc.
  - here also: architectural erosion

Other APL

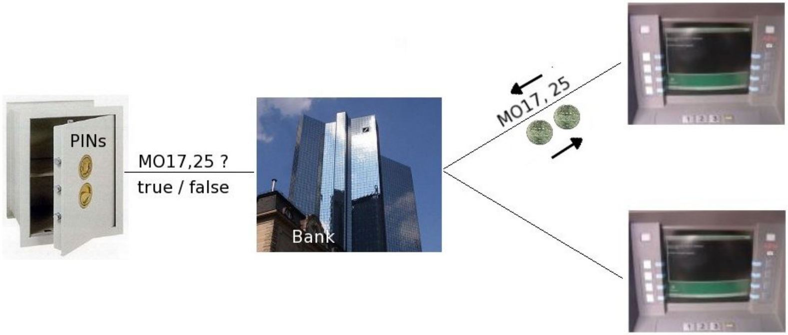
- **ArchJava** (D. Notkin et al., University of Washington)

Comparison of ArchJava and JAVA/A

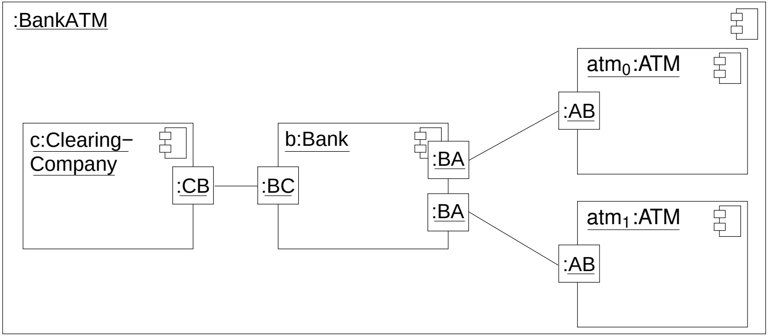
	components	*ports*	*configurations*	encapsulation
JAVA/A	yes	yes	explicit	yes
ArchJava	yes	yes	implicit	partial

	behavioural modeling	distributed applications	asynchronous communication
JAVA/A	yes	yes	yes
ArchJava	no	no	no

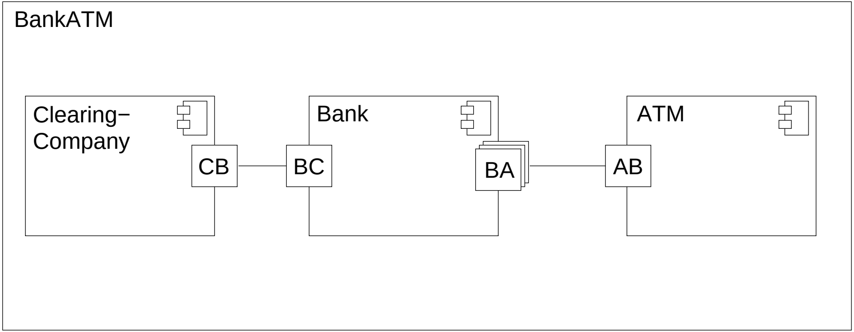
Example Bank–ATM (informal)



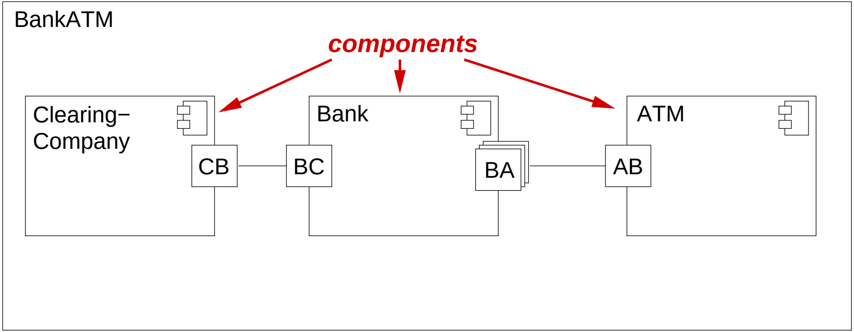
Example Bank–ATM: Configuration



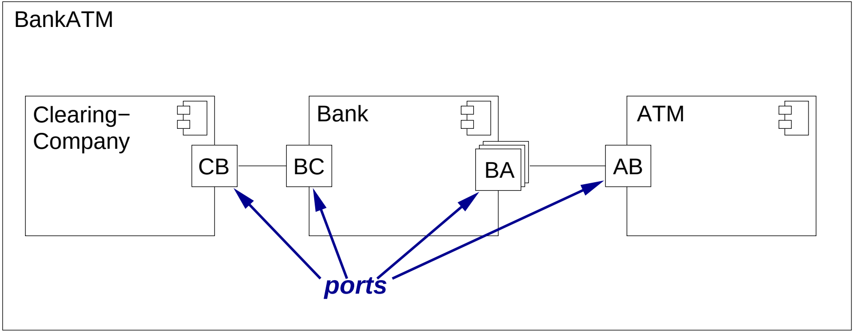
Example Bank–ATM: UML 2.0 Component Diagram



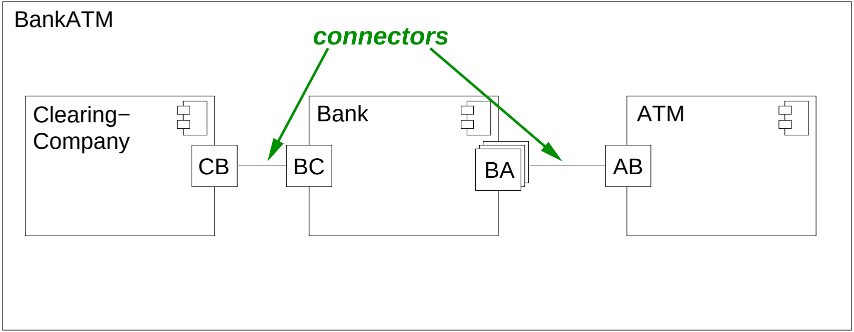
Example Bank–ATM: UML 2.0 Component Diagram



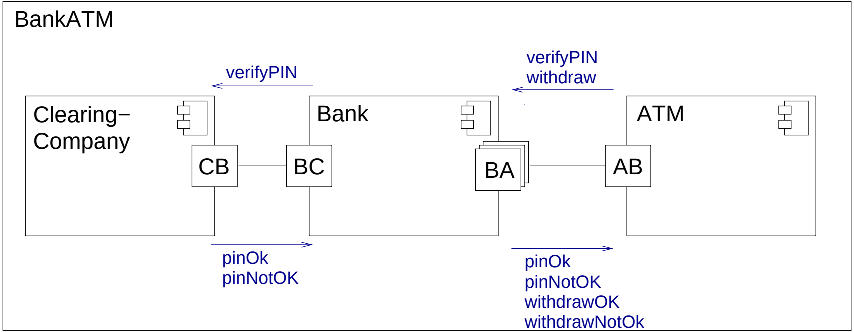
Example Bank–ATM: UML 2.0 Component Diagram



Example Bank–ATM: UML 2.0 Component Diagram



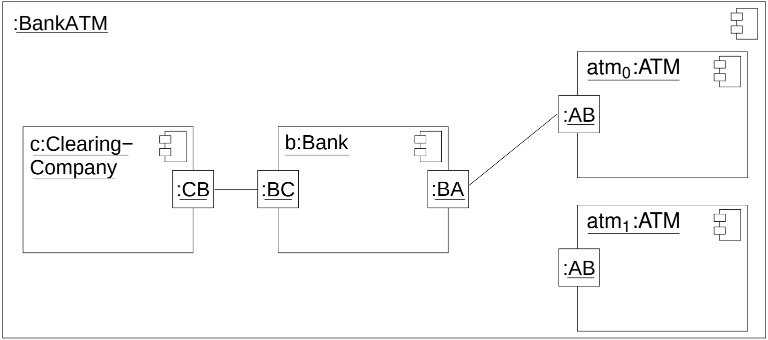
Example Bank–ATM: UML 2.0 Component Diagram



Example Bank–ATM: Dynamic Reconfiguration

Connecting components on demand

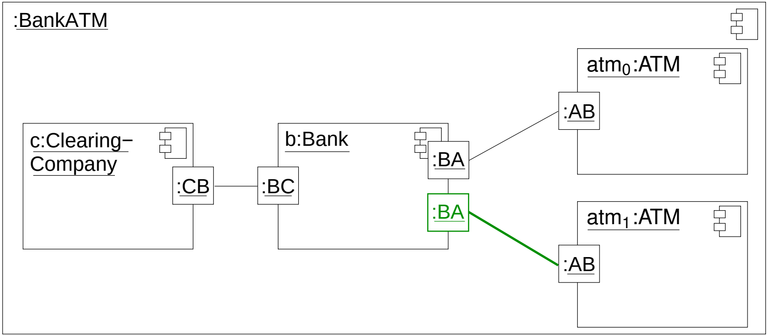
- ATMs are very often idle
- build up online connections only if necessary



Example Bank–ATM: Dynamic Reconfiguration

Connecting components on demand

- ATMs are very often idle
- build up online connections only if necessary



JAVA/A: An Architectural Programming Language (I)

Characteristics of JAVA/A

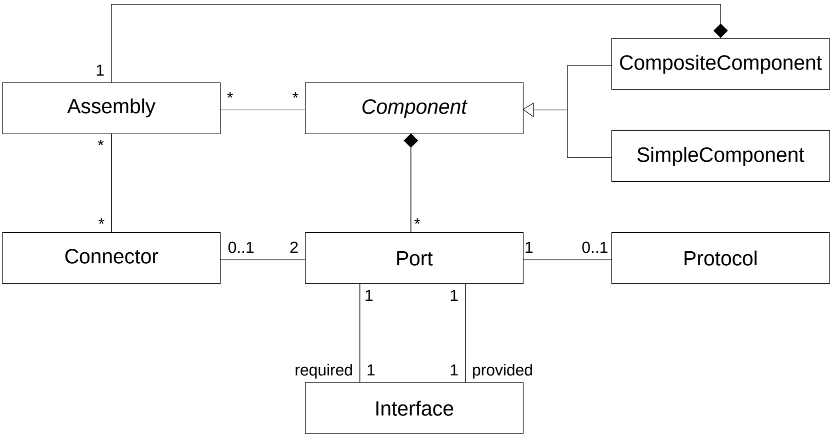
- components with ports (**strong encapsulation**)
- ports with **provided** and **required** interfaces, port **protocols**
- **hierarchical** composition
- **reconfiguration** at runtime
- **loose coupling** (independent deployment)
- **model checking support** for connectors (between ports)
- predefined **connector types** (local and distributed)

JAVA/A: An Architectural Programming Language (II)

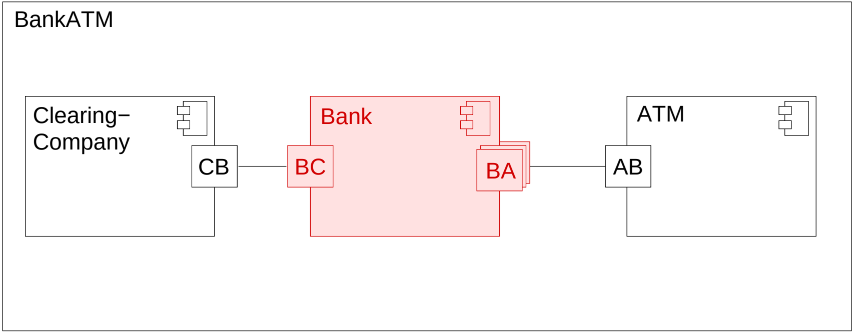
Current JAVA/A tool support

- compiler **jaac**
- Component Composition Platform (**CCP**):
  - building and editing configurations and components
- **coordination** and **management** framework
  - start and stop distributed applications

JAVA/A: Component Meta-Model



Example Bank–ATM: Implementation using JAVA/A



Example Bank–ATM: Implementation using JAVA/A

```
simple component Bank {
    Queue pending = new LinkedList();
    BA current = null;
    Set verifieds = new HashSet();
    Map balance = new HashMap();

    dynamic port BA {
        provided {
            signal verifyPIN(IBAN iban, int pin);
            signal withdraw(IBAN iban, Money amount);
        }
        required {
            void pinOk(); void pinNotOk();
            void withdrawOk(); void withdrawNotOk();
        }
    }
```

Example Bank–ATM: Implementation using JAVA/A

```
port BC {
    provided {
        void pinOk(); void pinNotOk();
    }
    required {
        void verifyPIN(IBAN iban, int pin);
    }
}
```

Example Bank–ATM: Implementation using JAVA/A

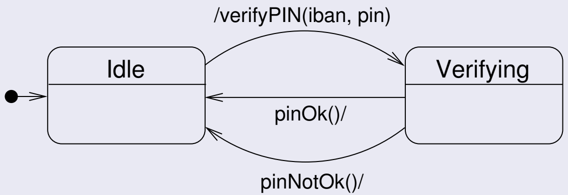
```
signal verifyPIN(BA incoming, IBAN iban, int pin)
            implements BA.verifyPIN(IBAN, int) {
   pending.offer(new Object[]{incoming,iban,pin});
   // wait until current == null
   Object[] request = (Object[])pending.poll();
   current = (BA)request[0];
   BC.verifyPIN((IBAN)request[1],
               ((Integer)request[2]).intValue());
}

void pinOk() implements BC.pinOk() {
   verifieds.add(current);
   current.pinOk();
   current = null;
   // notification that current == null
}
```

Specification of Port BC

- Ports are specified with UML 2.0 protocol state machines.

Specification of Port BC (Bank)



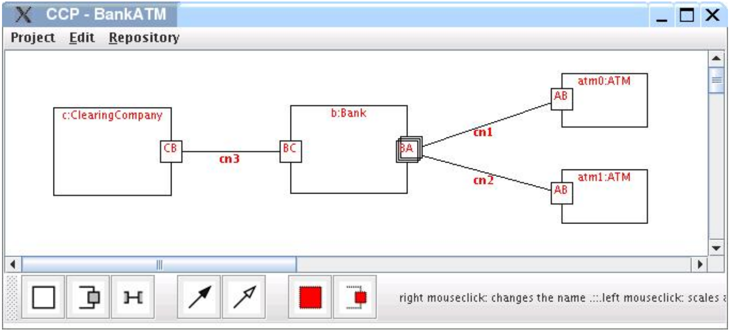
Example Bank–ATM: Configuration in JAVA/A

```
composite component BankATM {
  assembly {
    components { ATM, Bank, ClearingCompany }
    connectors { (ATM.AB, Bank.BA);
                 (Bank.BC, ClearingCompany.CB) }
    initial configuration {
      ATM atm0 = new ATM();
      ATM atm1 = new ATM();
      Bank bank = new Bank();
      ClearingCompany cc =
        new ClearingCompany();
      Connector cn0 = new Connector();
      cn0.connect(atm0.AB, bank.BA);
      Connector cn1 = new Connector();
      cn1.connect(atm1.AB, bank.BA);
      Connector cn2 = new Connector();
      cn2.connect(bank.BC, cc.CB); } } }
```

Example Bank–ATM: Reconfiguration in JAVA/A

```
try {
    Component bank = componentLookUp(this, "Bank");
    Port ba = bank.getPort("BA");
    ConnectionRequest cr =
        new ConnectionRequest(this,
            this, AB, bank, ba, new Connector());
    reconfigurationRequest(cr);
}
catch (ReconfigurationException e) {
    ...
}
```

Seamless Development



CCP: Component Composition Platform

- code generation
- round-trip engineering
- model checker integration for architectural analysis

A Model for AP: Overview (I)

Ports

- Signatures
  - provided interface $I = (\Sigma^{\mathrm{pro}}, Op^{\mathrm{pro}})$
  - required interface $O = (\Sigma^{\mathrm{req}}, Op^{\mathrm{req}})$
- Models: labelled transition systems
  - transitions: labelled by operation calls of provided and required operations $op(v)/$, $/op(v)$

A Model for AP: Overview (II)

Components

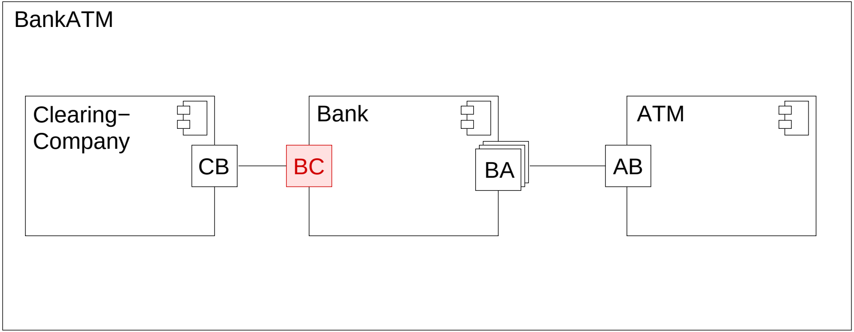
- Signatures
  - algebraic signature $\Sigma = (S, F)$ for internal states
  - port declarations of the form $P : \Sigma_P$
- Models: labelled transition systems
  - states: algebras over the internal state signature
  - transitions: labelled by operation calls on port instances $p.op(v)/$, $/p.op(v)$

A Model for AP: Overview (III)

Assemblies

- Signatures
  - algebraic signature for internal state
  - component declarations of the form $C : \Sigma_C$
  - connector declarations of the form $Con : \Sigma_{Con}$
- Models: labelled transition systems
  - states: algebras over the internal state signature (of the assembly)
  - transitions: labelled by operation calls on port instances of component instances with synchronization on connected ports $(c_1.p_1, c_2.p_2).op(v)$

Example: Signature Σ_{BC} for Port BC (Bank)



Example: Signature Σ_{BC} for Port BC (Bank)

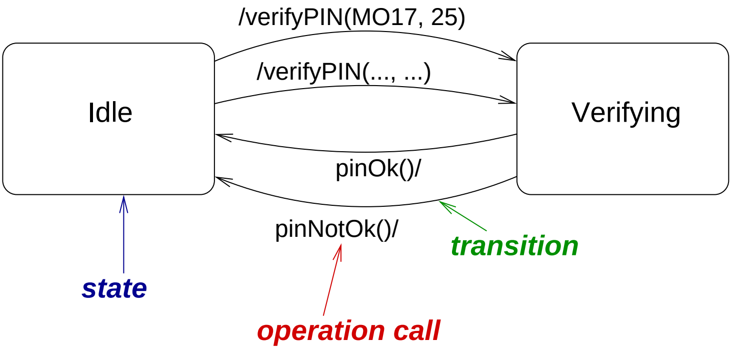
Provided Interface I_{BC}

- **sorts & funs:** $\emptyset$
- **operations:** `pinOk()`, `pinNotOk()`

Required Interface O_{BC}

- **sorts:** `int`, `IBAN`
- **funs:** $\mathcal{F}_{\texttt{int}}, \ldots$
- **operations:** `verifyPIN(iban: IBAN, pin: int)`

Example: A Model for Port BC (Bank)



Example: Signature Σ_{Bank} for Component Bank

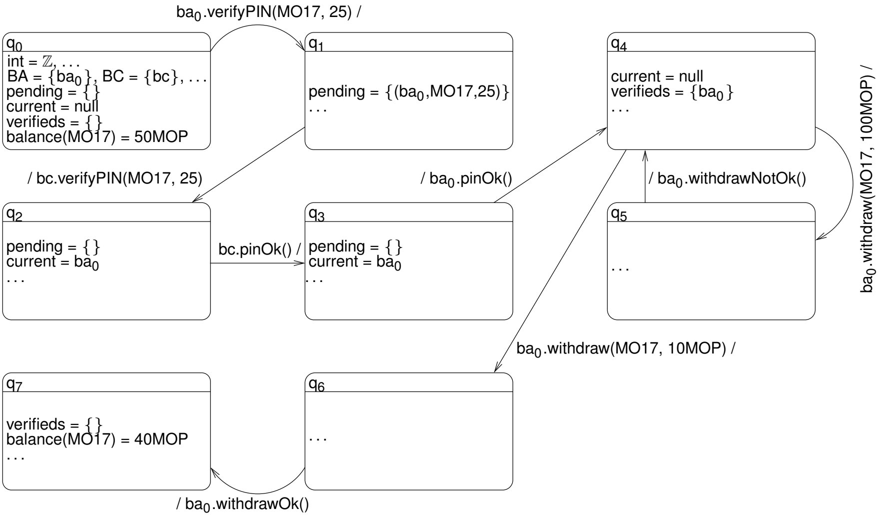
Port declarations

- BA : Σ_{BA}, BC : Σ_{BC}

Internal signature $\Sigma^{\mathrm{int}}_{Bank}$

- **sorts:** BA, BC, Queue, Set, Map, int, IBAN, Money
- **funs:**
  pending: → Queue,
  current: → BA,
  verifieds: → Set,
  balance: → Map,
  $\mathcal{F}_{\mathtt{int}}$, ...

Example: Part of a Model for Component Bank



Relationship between Ports and Components

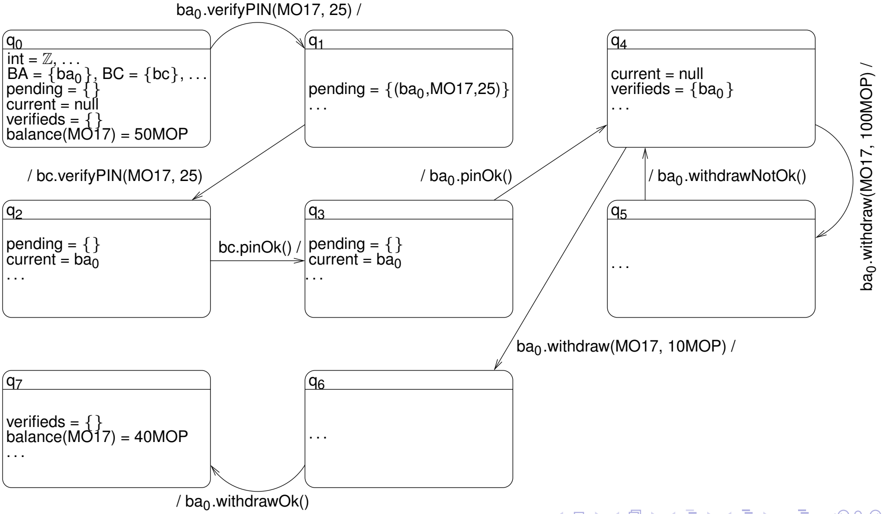
Component – Port

- a component has to **implement** all its ports
- ports can be dynamically **created**
- ports can be dynamically **destroyed**

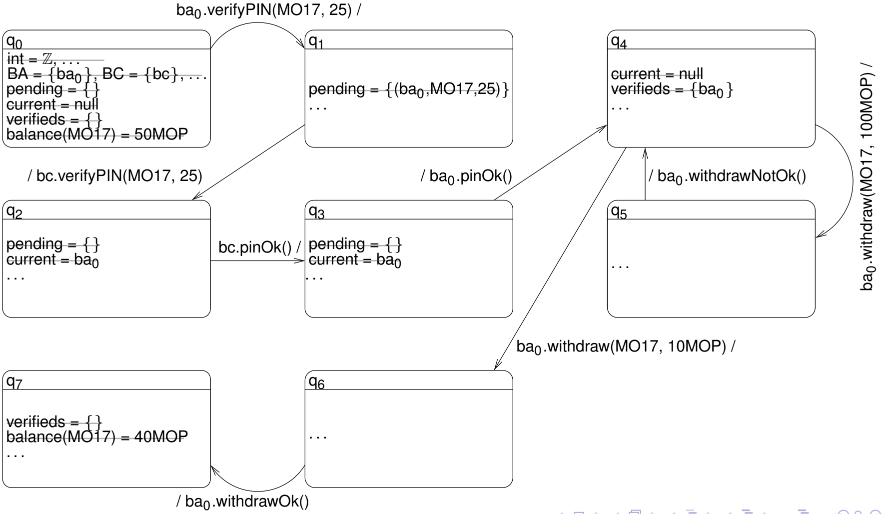
Implementation Correctness

- reduction of a component model to a port instance must be a model of the port (up to observ. equiv.)

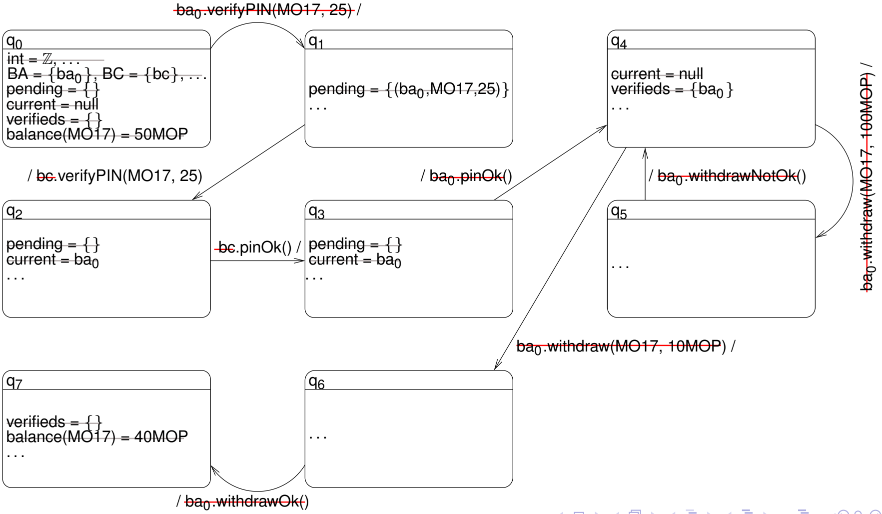
Reduct from the Model of Bank to Port Instance bc



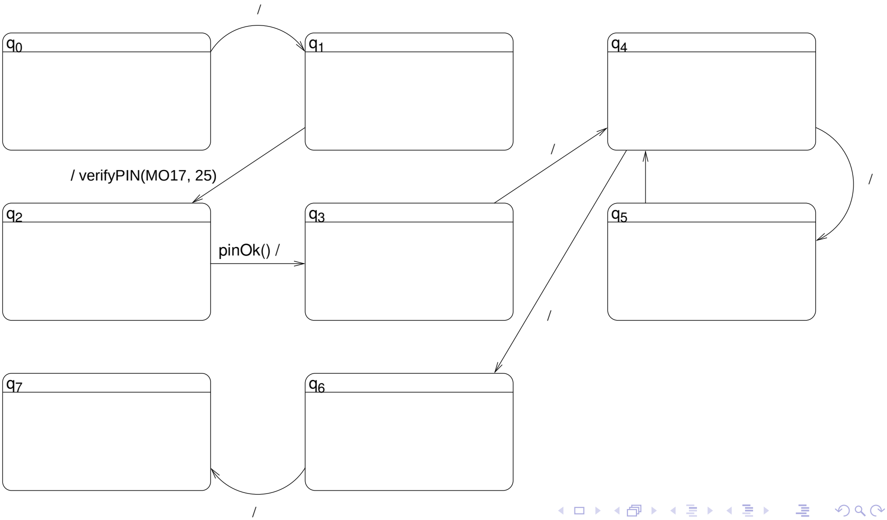
Reduct from the Model of Bank to Port Instance bc



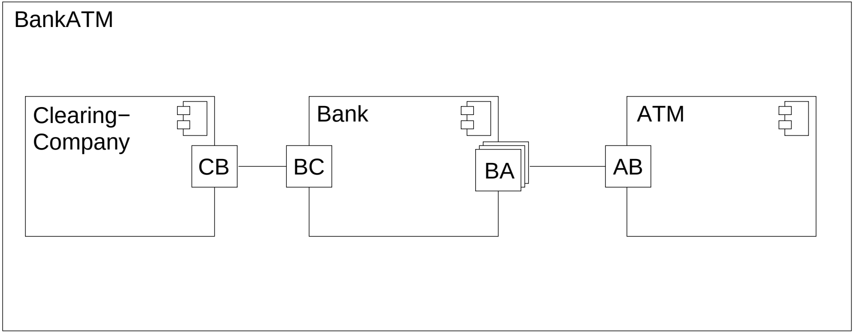
Reduct from the Model of Bank to Port Instance bc



Reduct from the Model of Bank to Port Instance bc



Example: Signature $\Sigma_{BankATM}$ for Assembly of Bank–ATM



Example: Signature $\Sigma_{BankATM}$ for Assembly of Bank–ATM

Component Declarations

ATM : Σ_{ATM}, Bank : Σ_{Bank}, ClearingCompany : $\Sigma_{ClearingCompany}$

Connector Declarations

ABBA : (AB, BA), BCCB : (BC, CB)

Internal Signature

- **sorts:** `ATM, Bank, ClearingCompany,`
  `AB, BA, BC, CB, ABBA, BCCB, ...`
- **funs:** `pending: Bank` $\rightarrow$ `Queue,`
  `current: Bank` $\rightarrow$ `BA,`
  `verifieds: Bank` $\rightarrow$ `Set,`
  `balance: Bank` $\rightarrow$ `Map, ...`

Example: Part of a Model for Assembly of Bank–ATM (I)

Component instances

$\mathrm{ATM} = \{\mathrm{atm}_0, \mathrm{atm}_1\}$, $\mathrm{Bank} = \{\mathrm{b}\}$, $\mathrm{ClearingCompany} = \{\mathrm{c}\}$

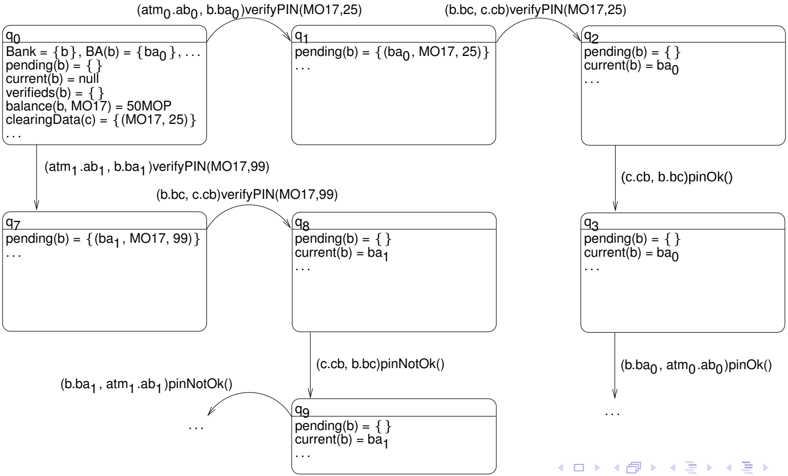
Port instances

$\mathrm{AB}(\mathrm{atm}_0) = \{\mathrm{ab}_0\}$, $\mathrm{AB}(\mathrm{atm}_1) = \{\mathrm{ab}_1\}$,
$\mathrm{BA}(\mathrm{b}) = \{\mathrm{ba}_0, \mathrm{ba}_1\}$, $\mathrm{BC}(\mathrm{b}) = \{\mathrm{bc}\}$, $\mathrm{CB}(\mathrm{b}) = \{\mathrm{cb}\}$

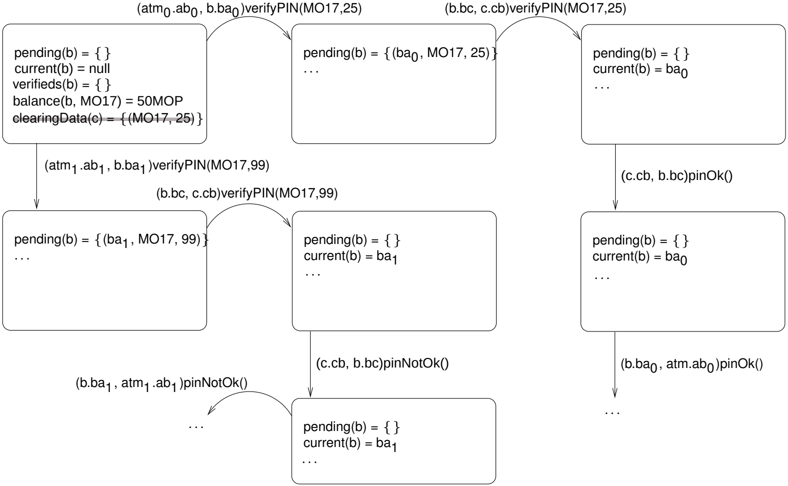
Connector instances

$\mathrm{ABBA} = \{(\mathrm{ab}_0, \mathrm{ba}_0), (\mathrm{ab}_1, \mathrm{ba}_1)\}$, $\mathrm{BCCB} = \{(\mathrm{bc}, \mathrm{cb})\}$

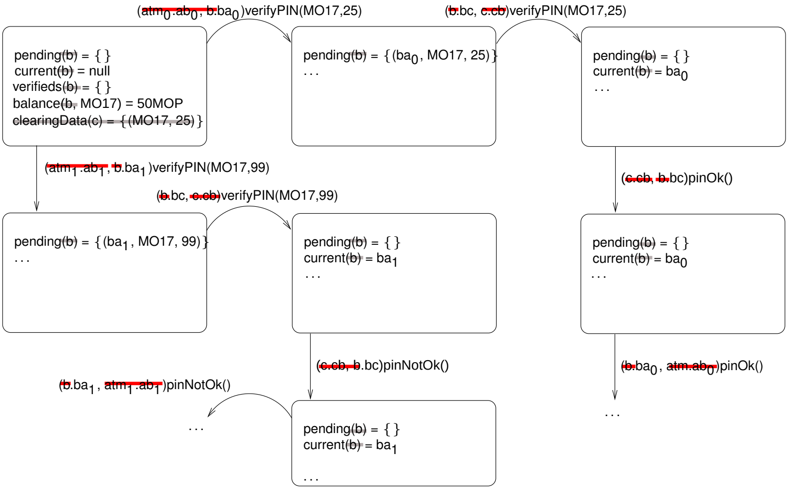
Example: Part of a Model for Assembly of Bank–ATM (II)



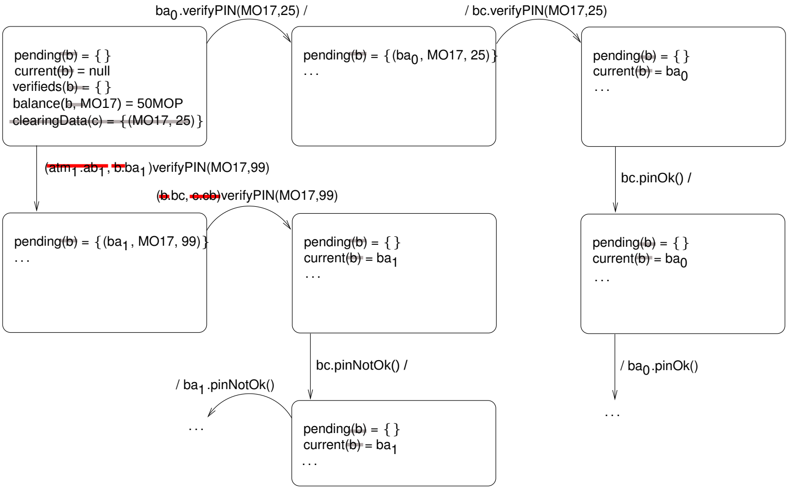
Reduct from Model Bank–ATM to Component b



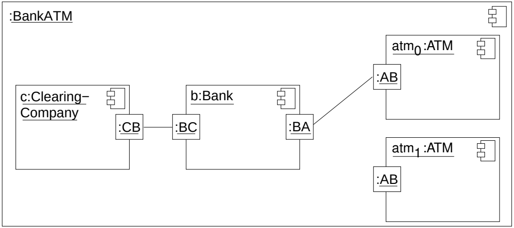
Reduct from Model Bank–ATM to Component b



Reduct from Model Bank–ATM to Component b



Example Bank–ATM: Reconfiguration



Port instances:

$AB(atm_0) = \{ab_0\}$,
$AB(atm_1) = \{ab_1\}$,
$BA(b) = \{ba_0\}$

Connector instances:

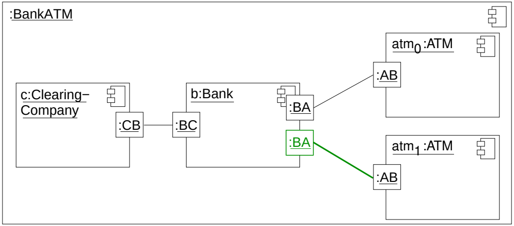
$ABBA = \{(ab_0, ba_0)\}$

Port instances:

$AB(atm_0) = \{ab_0\}$,
$AB(atm_1) = \{ab_1\}$,
$BA(b) = \{ba_0, ba_1\}$

Connector instances:

$ABBA = \{(ab_0, ba_0), (ab_1, ba_1)\}$



Checking Connectors for JAVA/A

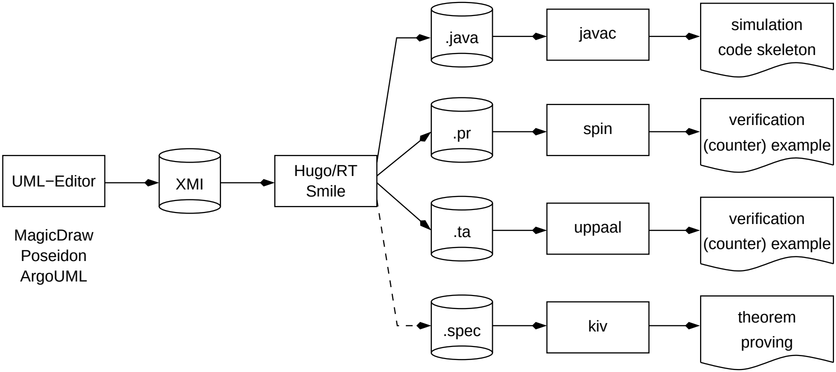
What is checked?

- Syntactically: interface conformance
- Semantically: absence of deadlocks

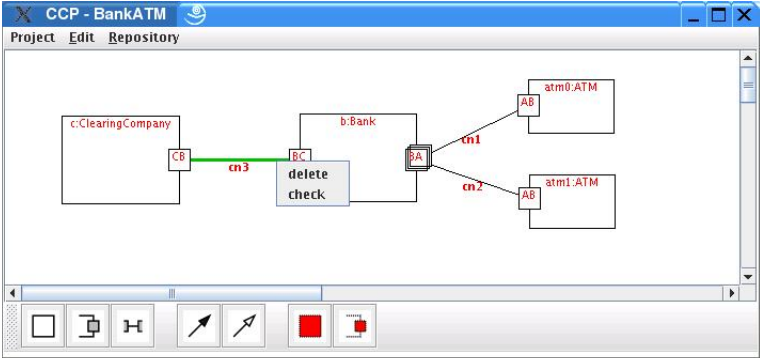
How is it checked?

- interface conformance: by the compiler
- absence of deadlocks: model checking using Hugo/RT
    - for finite state systems only . . .

Hugo/RT


UML−Editor
MagicDraw
Poseidon
ArgoUML
XMI
Hugo/RT
Smile
.java
.pr
.ta
.spec
javac
spin
uppaal
kiv
simulation
code skeleton
verification
(counter) example
verification
(counter) example
theorem
proving

Applying Hugo/RT in CCP



Model checking connnectors

- “one-click”
- no model checking knowledge necessary

Correctness of (Static) Configurations

Theorem

Let Ξ_1 and Ξ_2 be port specifications. Let Γ be a configuration containing two component instances c_1 and c_2, such that c_1 has a port instance p_1 satisfying Ξ_1 and c_2 has a port instance p_2 satisfying Ξ_2 and p_1 and p_2 are connected.

$$\Xi_1 \parallel \Xi_2 \models \neg\delta \quad \Rightarrow \quad \Gamma \models \neg\delta$$

Conclusions and Future Work

Architectural Programming with Java/A

- bridging the gap between software architecture and programming languages
- based on an algebraic component model

Future Work

- extensions: ”explicit” ports of composite components, n-ary connectors, shared components, . . .
- specification framework: for reconfigurations, internal component behavior, . . .
- proof techniques (for refinements, . . .)
- black box and glass box semantics