Architecture Evolution

Alexander Serebrenik

TU/e Technische Universiteit **Eindhoven** University of Technology

Where innovation starts

Assignments: Reminder

- **Assignment 1:**
  - Requirements
  - Deadline: February 19, 2016 23:59
    - This happens to be *today*
    - Do not wait till 23:58…

- **Assignment 2:**
  - Architecture reconstruction
  - Deadline: March 9, 2016 23:59

Sources

MONOGRAPHS IN COMPUTER SCIENCE

REVERSE ENGINEERING OF OBJECT ORIENTED CODE

Paolo Tonella
Alessandra Potrich

Ch. 4-5

Springer

Tom Mens
Serge Demeyer (Eds.)

Software Evolution

Ch. 10

Springer

TU/e Technische Universiteit
Eindhoven
University of Technology
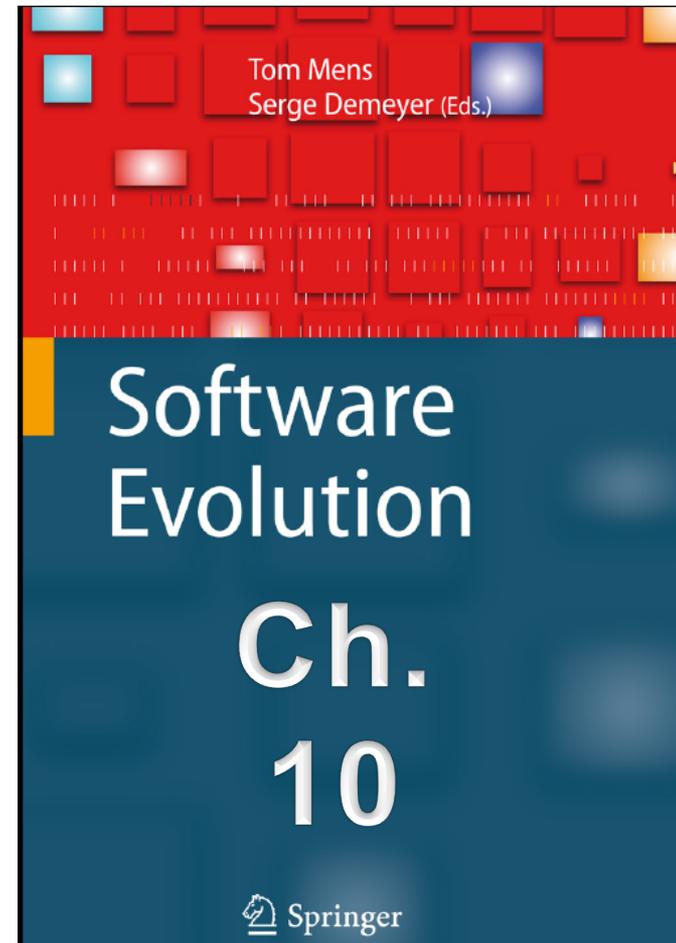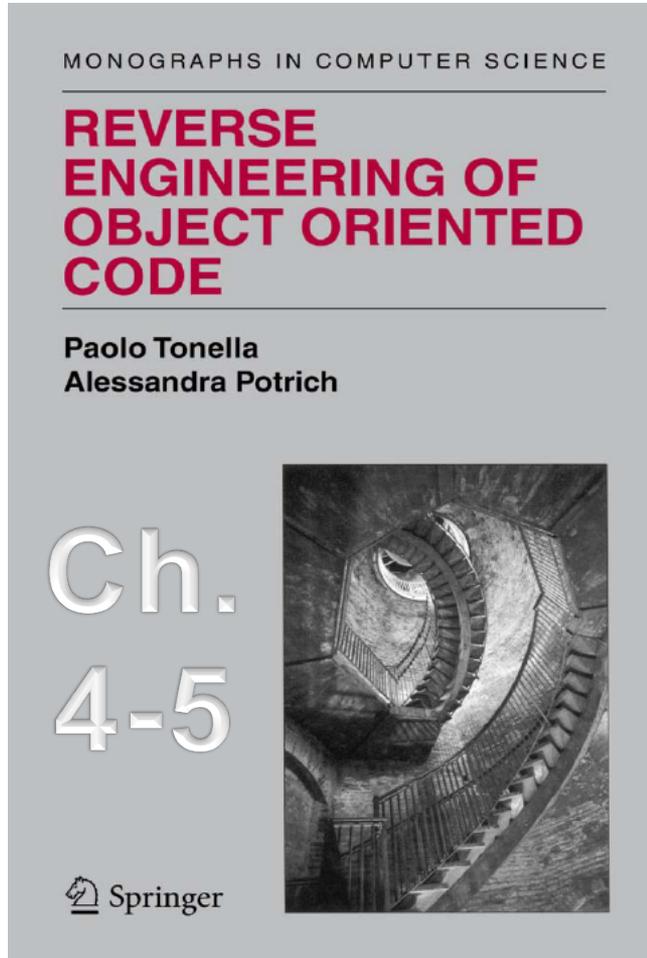
Part 1: Where are we now?

- **Last time: architecture**
  - **Structure (class and package diagrams)**

- **Today: behaviour**
  - **Sequence diagrams**
  - **State diagrams (State machines)**

Static vs. Dynamic Analysis

- **Dynamic (execution)**
  - \+ **More precise: at run-time you know everything**
  - \- **Requires the system to be executable**
  - \- **Limited to test-cases**
- **Static (no execution required)**
  - \- **Less precise (approximate but conservative)**
  - \+ **The system may be incomplete**
  - \+ **All possible executions can be considered**

Static vs. Dynamic Analysis

- **Dynamic (execution)**
  - \+ **More precise: at run-time you know everything**
  - \- **Requires the system to be executable**
  - \- **Limited to test-cases**
- **Static (no execution required)**
  - \- **Less precise (approximate but conservative)**
  - \+ **The system may be incomplete**
  - \+ **All possible executions can be considered**

		Object of the analysis	
		Structure	**Behaviour**
Analysis technique	**Static**	Last time	Today
	Dynamic		

Static analysis of behavioural models

| Library1: Library | Loan1: Loan | InternalUser1:InternalUser, User1: User | Book1: Book, TR1:TR, Journal1:Journal |

getUser →

getDocument →

addLoan →

addLoan →
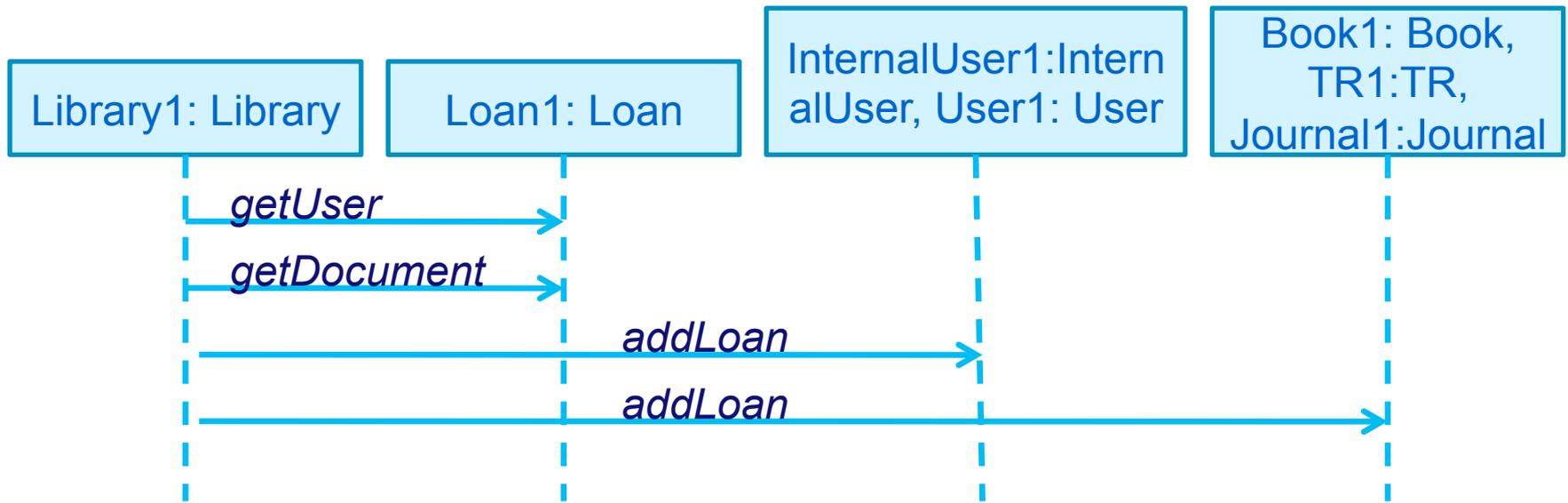
- **Columns**
  - **Containers**
  - **Objects vs. classes**
    - **Objects: more precise**
    - **Objects: Too many/too few?**

- **Method invocations**
  - **Resolving the calls**
 - **Invocation order**

Recapitulation: Object Flow Graph

- **Vertices: variables, fields, method parameters**
- **x = y**

- **x = y.m(a_1, …, a_k)**
  - **Method m(f_1, …, f_k)**
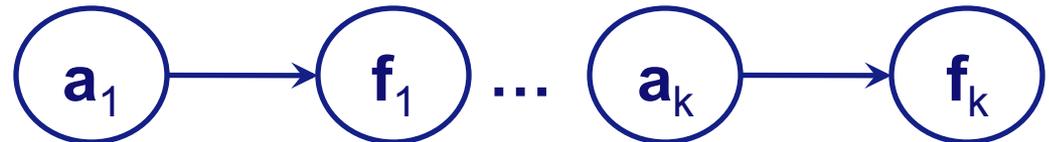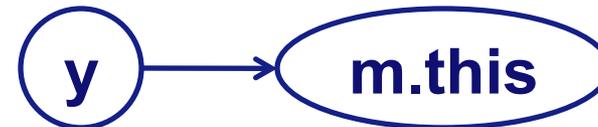
- **x = new c(a_1, …, a_k)**
  - **cs – constructor of c**

TU/e Technische Universiteit
Eindhoven
University of Technology
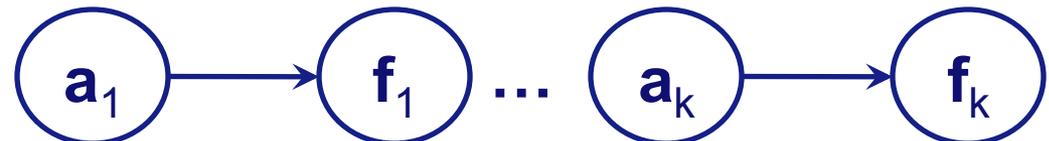
Objects vs. Classes

```
class BT {
    BTNode root;
    public void build() {
  1   root = new BTNode();
        BTNode curNode = root;
        while(…) {

            …
    2   curNode.addLeft(new BTNode());
            …
    3   curNode.addRight(new BTNode());
        }
    }
}
```

```
class BTNode {
    BTNode left, right;
    public void addLeft(BTNode n) {
        left = n;
    }
    public void addRight(BTNode n) {
        right = n;
    }
}
```

Tonella, Potrich 2005
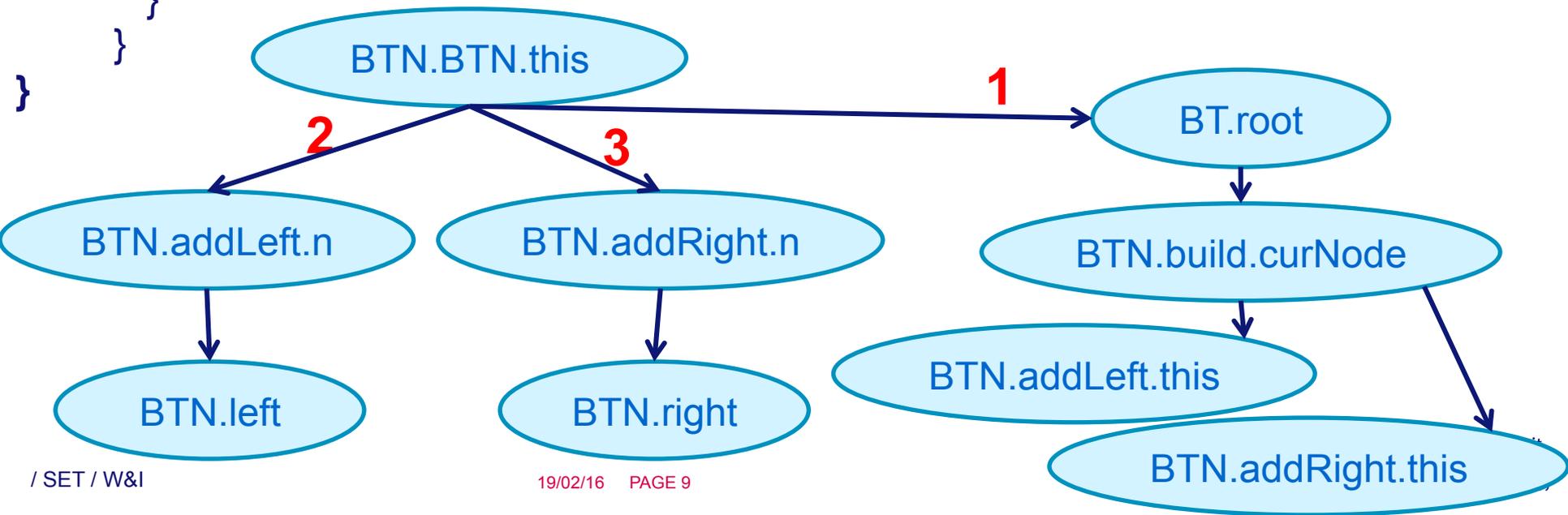
Solution

- **Generate**
  - **Increment the counter for every "new BTNode"**
- **Propagate this information through the OFG**

BTN.BTN.this

1 BT.root

2 gen(BTN.add-Left.n) = {BTN2}

3 gen(BTNode.add-Right.n) = {BTN3}

gen(BT.root) = {BTN1}

BTN.addLeft.n

BTN.build.curNode

out(BTN.left) = {BTN2}

BTN.addRight.n

out(BT.root) = {BTN1}

out(BTN.left) = {BTN3}

BTN.addLeft.this

BTN.left

BTN.right

BTN.addRight.this
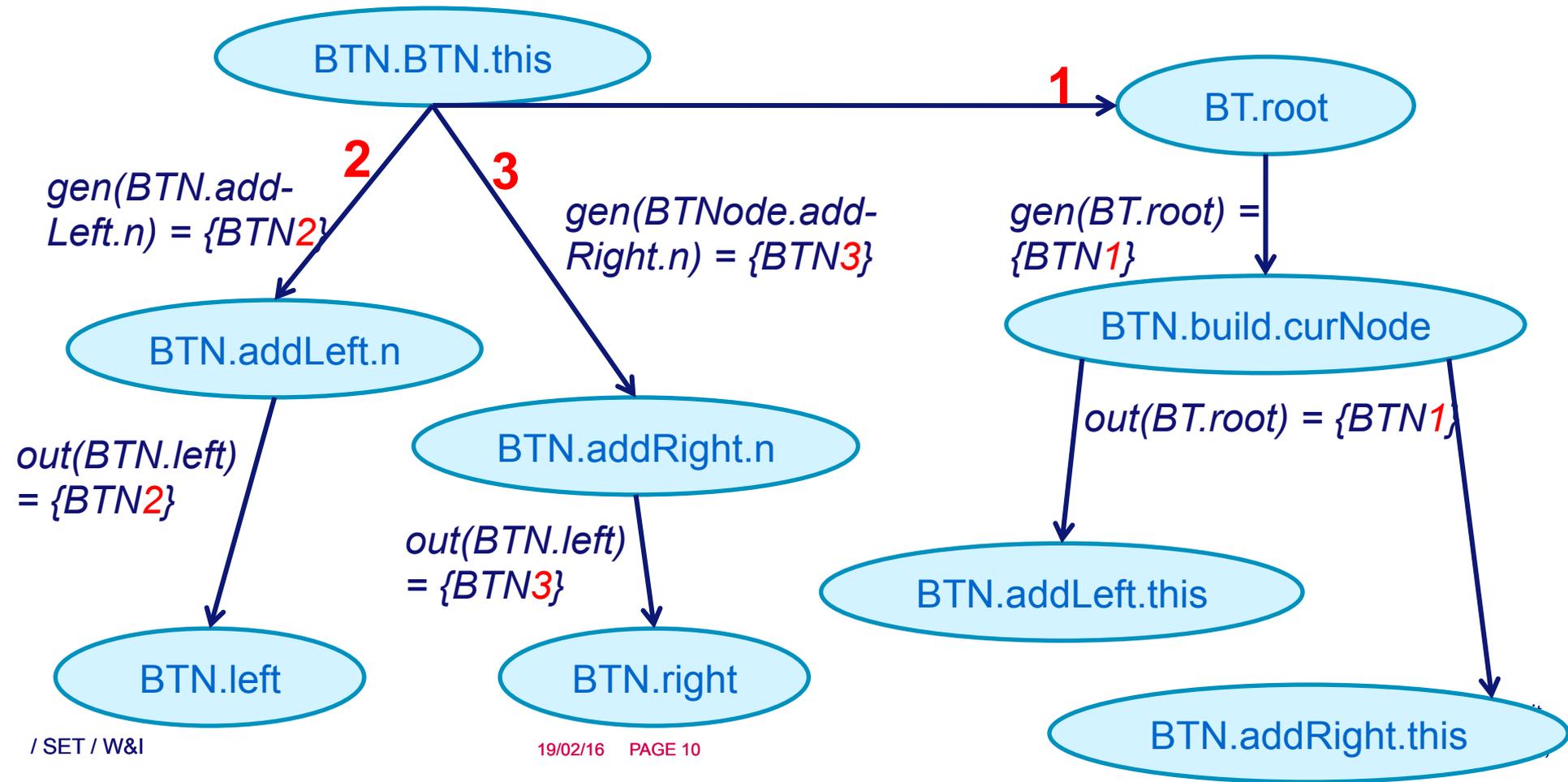
Objects vs. Classes

Tonella, Potrich 2005
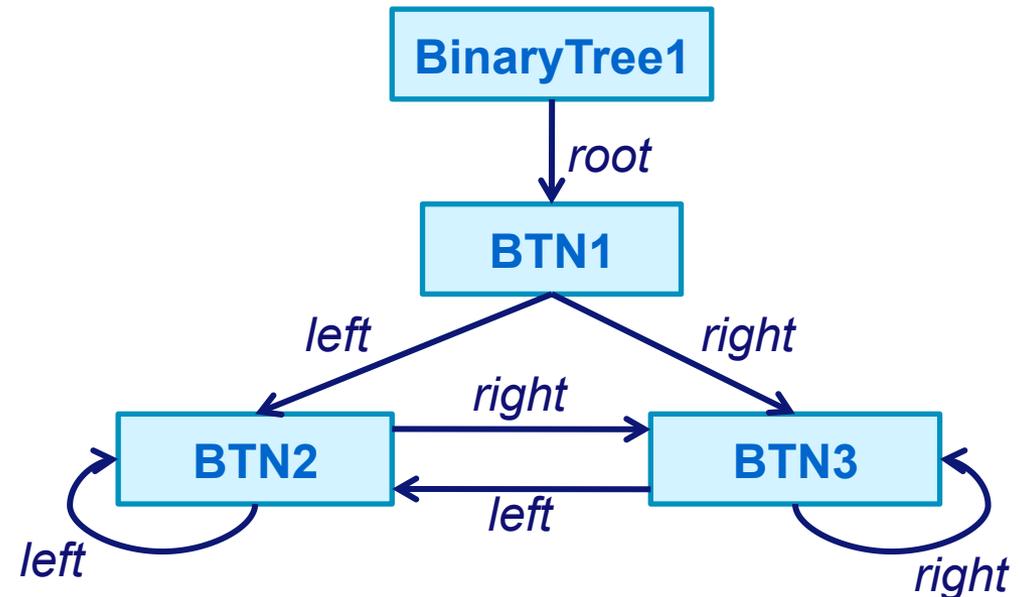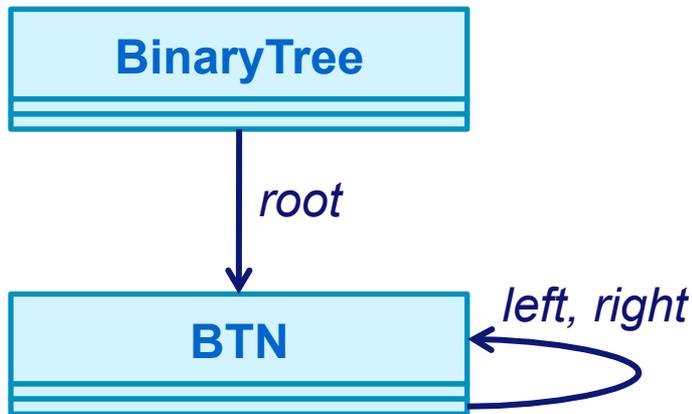
- **We can use the same technique for sequence diagrams!**

TU/e Technische Universiteit **Eindhoven** University of Technology

Next step: resolving method calls

```
class C {
    public m() {
        D.n()
    }
}
```

- **Intuitively, C.m() $\Rightarrow$ D**

- **In the OFG notation:**
  - **If D is a field: C.m.this $\Rightarrow$ C.D**
  - **Otherwise: C.m.this $\Rightarrow$ C.m.D**

- **But these are classes, we need objects!**
  - **If D is a field: out(C.m.this) $\Rightarrow$ out(C.D)**
  - **Otherwise: out(C.m.this) $\Rightarrow$ out(C.m.D)**

- **NB: out are sets (polymorphism)**

Out is a set!
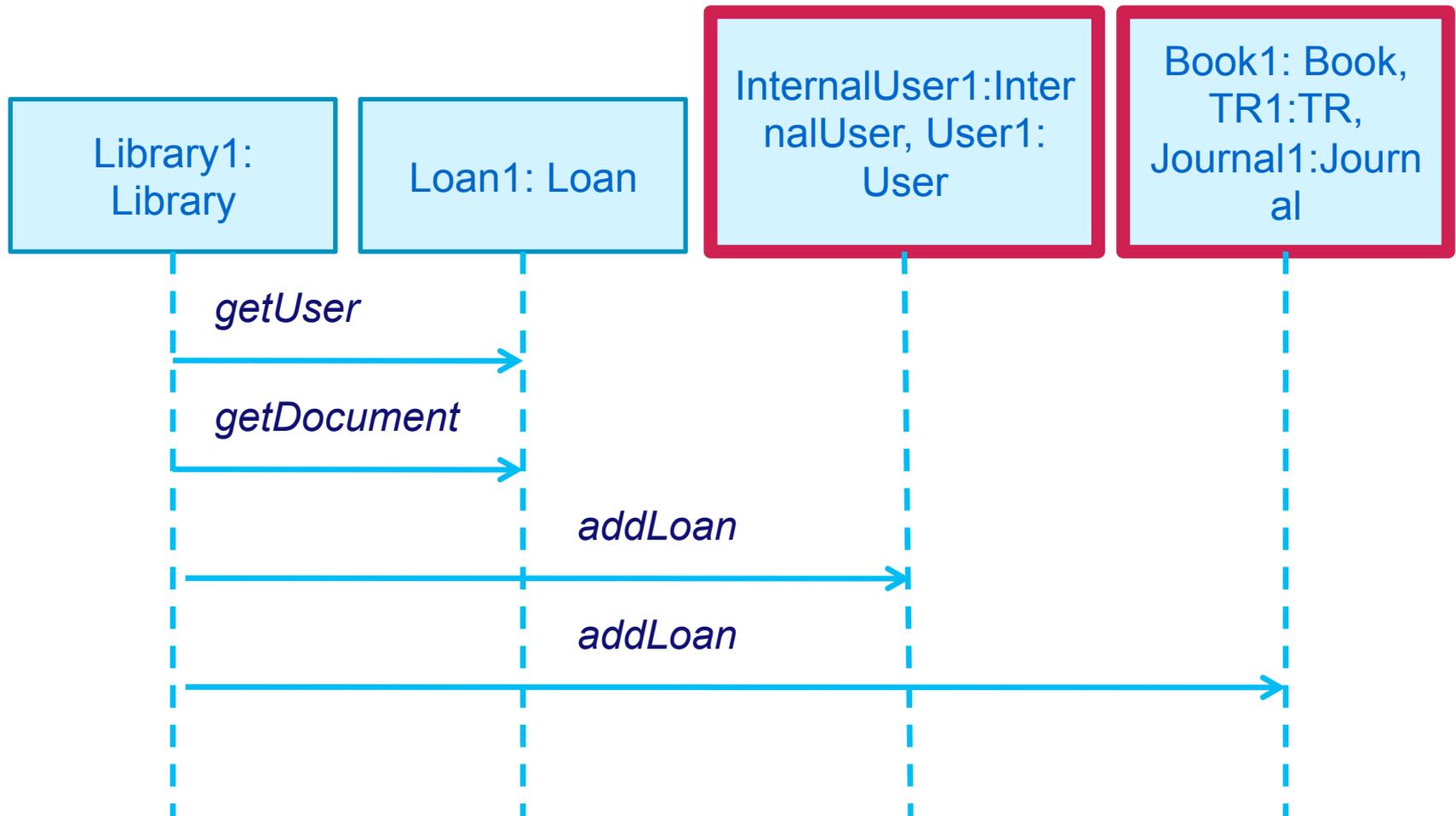
Invocation order

- **Traditional approach**
  - **Follow the code**
  - **UML 1.x: SD represents one execution path**
    - **Decide arbitrarily on if/while choices**
  - **UML 2.x: SD contains alterations, options and loops**
    - **Follow the code**

- **Can you think on Java programs that cannot be analysed in this way?**

Invocation order

- **Traditional approach**
  - **Follow the code**
  - **UML 1.x: SD represents one execution path**
    - **Decide arbitrarily on if/while choices**
  - **UML 2.x: SD contains alterations, options and loops**
    - **Follow the code**

- **Can you think on Java programs that cannot be analysed in this way?**
  - **Limitations of the approach**
    - **"Hidden" control flow rules: EJB interceptors**

Invocation order

- **Traditional approach**
  - **Follow the code**
  - **UML 1.x: SD represents one execution path**
    - **Decide arbitrarily on if/while choices**
  - **UML 2.x: SD contains alterations, options and loops**
    - **Follow the code**

- **Can you think on Java programs that cannot be analysed in this way?**
  - **Limitations of the approach**
    - **"Hidden" control flow rules: EJB interceptors**
  - **Limitations of the UML**
    - **Java/UML mismatches**

Interceptor

Logging, security, performance, …

```
public class importantClass {

    public String importantMethod() {
        …
    }
    …
}
```

Flexibility:
- **How?**
  - *XML file*
  - *Annotation*
- **At what level?**
  - *Bean class*
  - *Method*
- **Where?**
  - *Bean class*
  - *Superclass*
  - *Separate class*
  - *Injected bean*

Order of invocation

- **9 complex and interwined laws**
- **Spec is inconsistent with *GlassFish***

TU/e Technische Universiteit Eindhoven University of Technology

# Code Example: Enterprise Java Bean

```java
package ejb;
import ...

@Stateless
@Interceptors({Logger.class})
public class ProductFacade
extends EJBObject /* it may have an
interceptor */
implements ProductFacadeRemote {
    @PersistenceContext
    private EntityManager em;
        ...
    public Product find(Object id) {
        return em.find(Product.class, id);
    }

@Interceptors({Profiler.class})
public String productInfo(int id) {
    Product product = find(new Integer(id));
    if (product != null) {
        return " Description: "
            + product.getDescription();
    } else {
        return null;
    }
}
@AroundInvoke
@Override
protected Object logMethods(
    InvocationContext ctx)
        throws Exception {...}
}
```

TU/e
Technische Universiteit
Eindhoven
University of Technology

Example (cont.): XML deployment descriptor

```xml
<assembly-descriptor>
<!-- Default interceptor -->
   <interceptor-binding>
      <ejb-name> * </ejb-name>
      <interceptor-class> interceptor.Login </interceptor-class>
   </interceptor-binding>
<!-- Method interceptor -->
   <interceptor-binding>
      <ejb-name> ProductFacade </ejb-name>
      <interceptor-class> interceptor.Auditor </interceptor-class>
     <exclude-class-interceptors> true
     </exclude-class-interceptors>
      <method>
         <method-name> productInfo </method-name>
      </method>
   </interceptor-binding>
</assembly-descriptor>
```

TU/e Technische Universiteit
Eindhoven
University of Technology

Reverse engineered sequence diagram
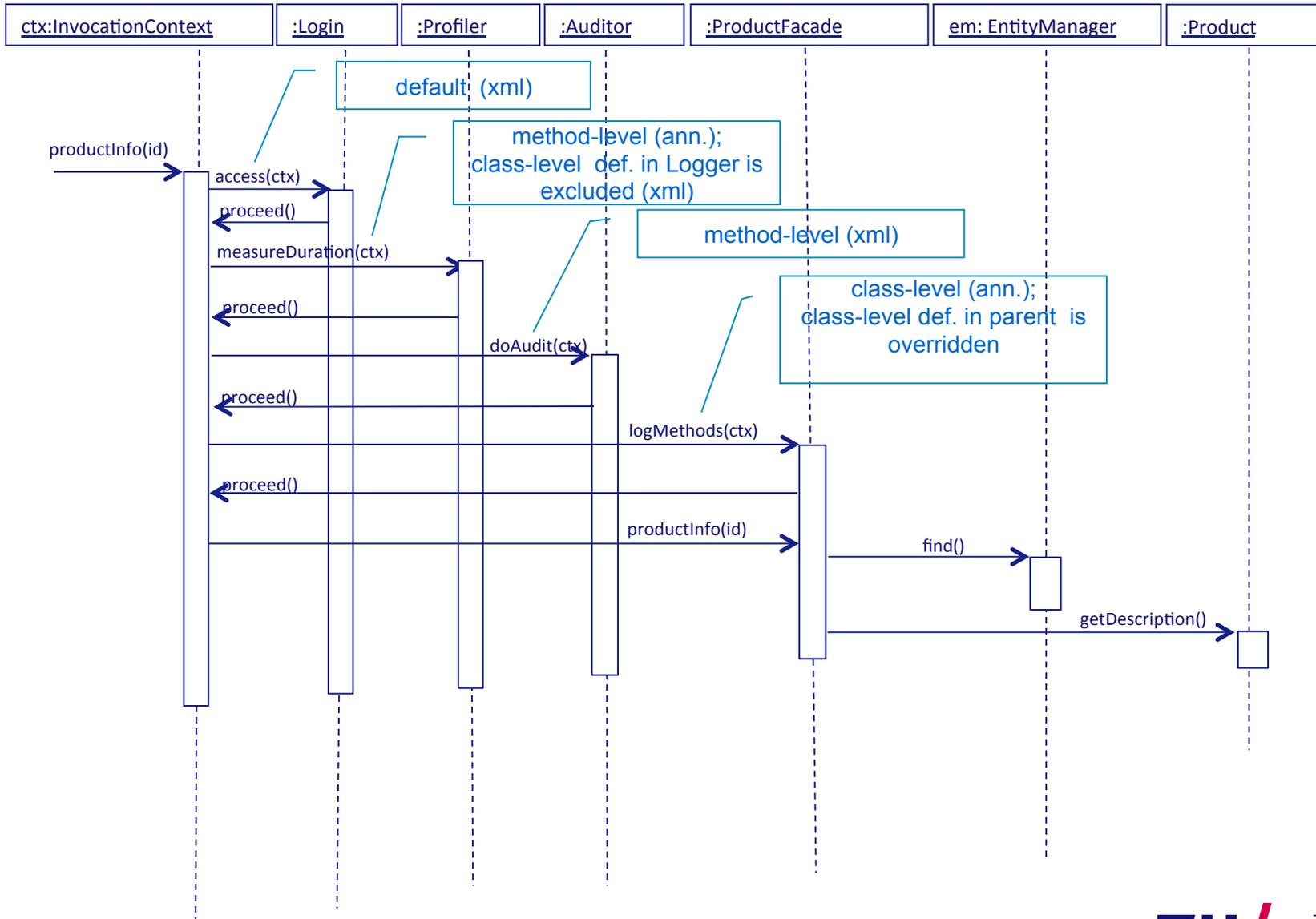
Limitations of UML: Java/UML mismatches

- **Java semantics does not match UML semantics**
- **UML break**
  - **if the selected interaction occurs, the enclosing interaction is abandoned**
- **Java break**

```
for (j = 0; j < arrayOfInts.length; j++) {
    if (arrayOfInts[j] == searchfor) {
        foundIt = true;
        break;
    }
}
```

Rountev, Volgin, Reddoch
PASTE'05

Technische Universiteit
Eindhoven
University of Technology

TU/e

Java/UML mismatches

- **Java semantics does not match UML semantics**
- **UML break**
  - **if the selected interaction occurs, the enclosing interaction is abandoned**
- **Java break**

```
search:
    for (i = 0; i < arrayOfInts.length; i++) {
        for (j = 0; j < arrayOfInts[i].length; j++) {
            if (arrayOfInts[i][j] == searchfor) {
                foundIt = true;
                break search;
            }
        }
    }
```

Rountev, Volgin, Reddoch
PASTE'05

TU/e Technische Universiteit
Eindhoven
University of Technology

Static analysis so far…

- **Architecture reconstructor has to decide**
  - **Classes or objects**

- **Methods**
  - **Resolve the method calls**
  - **Invocation order**
    - **May be complex (interceptors)**

TU/e Technische Universiteit
Eindhoven
University of Technology

Code

Instrumented code

Running system

Sequence diagram

Output
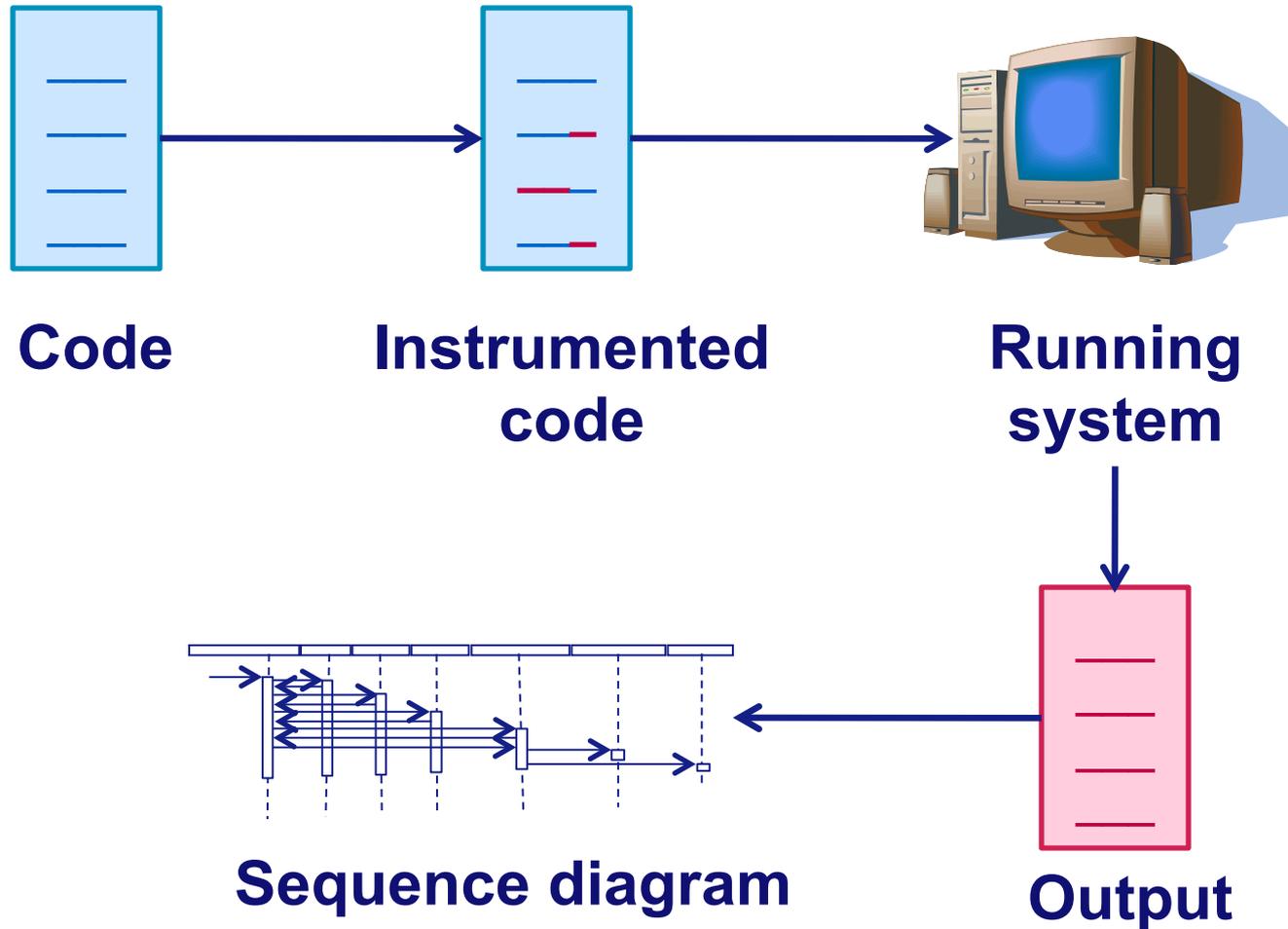
What would you like to instrument?

- **Source code**
  - **Maximal precision**

- **Compiled code**
  - **Example:**
    - **gcc -c -pg myfile.c**
    - **gcc -pg collatz.o**
  - **Works for any source code**
  - **Information is lost**

- **Execution environment (e.g., Java VM)**
  - **Idem**

TU/e Technische Universiteit
Eindhoven
University of Technology

How to instrument the code?

- **Classes: augment with the object identifier**
  - **Java: hashCode, rmi.ObjID, etc…**
- **Add to the trace**

<currObject,targetObject,methodName,timeStamp>

- **Trace ifs and loops**
  - **We cannot see the "other way"**
  - **We can see the constructs**
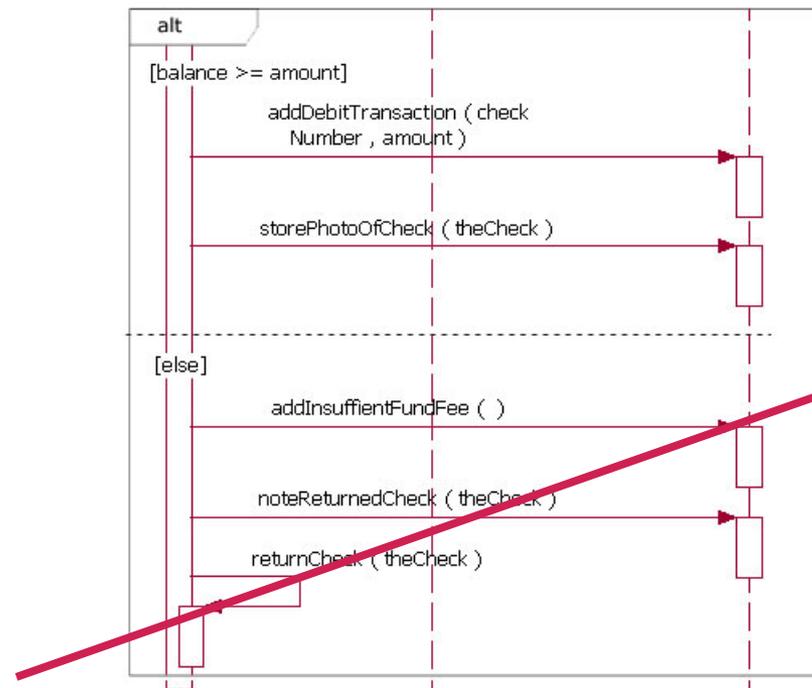
How to store the information?

- **XML with special tags**
  - **MXML, Columbus CAN, …**

- **Compact Trace Format [Hamou-Lhadj, Lethbridge 2004]**
  - **Common subtrees are represented only once**
    - **Similar to ATerms library (ASF+SDF, mCRL2,…)**

- **XES [Günther 2009]**
  - **Used for process mining**

The Size Explosion problem

- **One process in a printer component**
  - **129 calls from the main function**
  - **18 objects (1 per class) involved**
  - **Picture does not fit the slide**

- **Solutions**
  - **Smaller scale**
    - **"interesting" top-level functions**
    - **"interesting" calls in these functions**
    - **"interesting" target objects/classes**

  - **Different visualization**

Require a priori knowledge

Jerding, Stasko, Ball 1997

Message-flow diagram
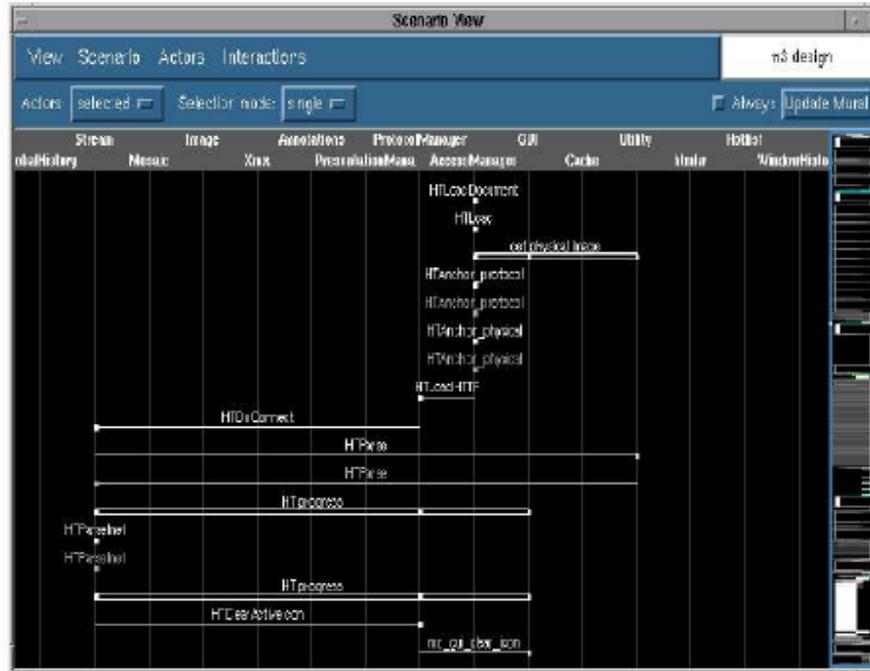
Information mural view: general overview in small

One can define interaction patterns and search for their appearances.

TU/e Technische Universiteit
Eindhoven
University of Technology

Better visualization (2)

Holten, 2009

Hierarchy of packages and classes

Arrow:
from green
to red

Repeated pattern and exceptional behaviour are clearly visible. Solves the blurring up problem by special visualization technique

Summary so far

- **Static and dynamic approaches to sequence diagram reconstruction**
  - **Static:**
    - **Objects or classes**
    - **Resolve the method calls**
    - **Invocation order**
  - **Dynamic:**
    - **What should be instrumented?**
    - **How should it be instrumented?**
    - **How should the information be stored?**
- **Still: size explosion problem**
  - **Limitation vs. Visualization**

State machines

- **States and transitions**

- **Usually describe behaviour within one class**

- **Either**
  - **Explicit**
    - **Developers intentionally encode states and transitions**
  - **Implicit**
- **Also useful for model checking.**

http://franck.fleurey.free.fr/VirtualMeeting/index.html

Technische Universiteit
Eindhoven
University of Technology
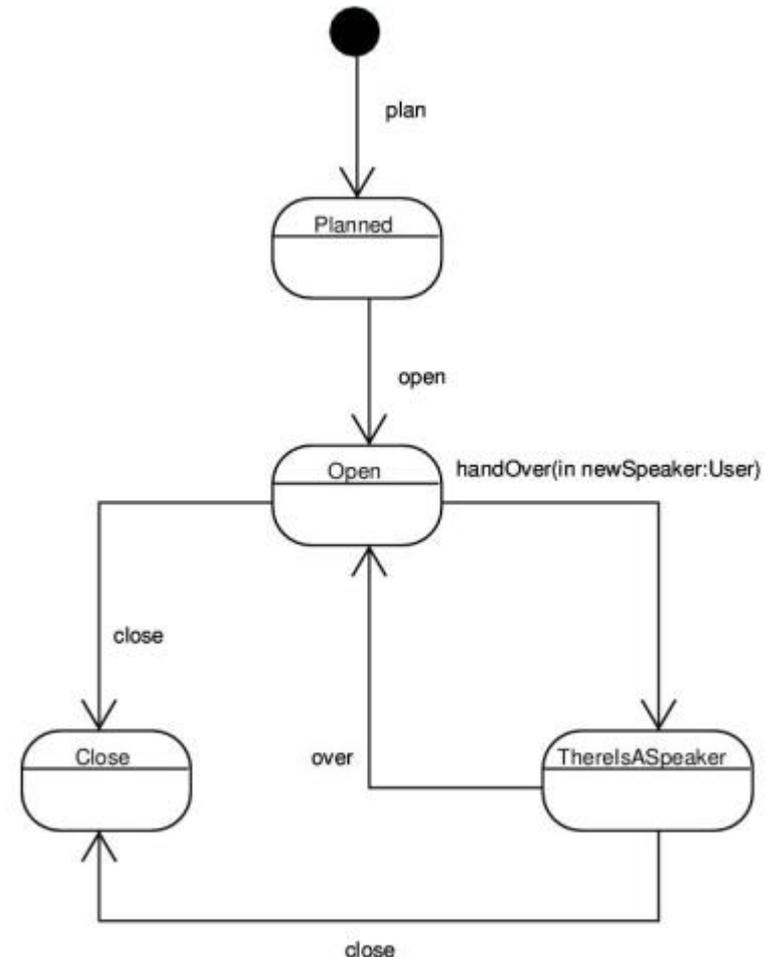
Explicit State Machine Patterns

- **Nested Choice Patterns**
  - **Switch/Case – Switch/Case**
  - **Switch/Case – If/Else**
  - **If/Else – Switch/Case**
  - **If/Else – If/Else**

```c
typedef struct {
  bool initialised;
  OBJ_STATE state;
} OBJ;
...
switch (obj->state) {
  case STATE_A:
    switch (event) {
      case EVENT1:
        obj->state = STATE_B;
        break;
      case EVENT2:
        obj->state = STATE_C;
        break;
    } break;
  case STATE_B:
    switch (event) {
      case EVENT1:
        obj->state = STATE_B;
        break;
      case EVENT2:
        obj->state = STATE_C;
        break;
    } break;
    ...
}
```

TU/e Technische Universiteit
Eindhoven
University of Technology

Explicit State Machine Patterns

- **Nested Choice Patterns**

- **Jump Tables**
  - **State = a pointer to a function pointer array (jump table)**
  - **Transitions = function in the jump table**

```c
typedef int (*Handler)(char *);

/* The functions */
int func0 ( char *event) {
    int state;
    state = …
    return state;
}
…
Handler jt[4] =
    {func0,
    func1,
    func2,
    func3};
…
int main () {
    int state;

    while(true) {
        state = jt[state]
(event);
    }
}
```

Technische Universiteit
Eindhoven
University of Technology

Explicit State Machine Patterns

- **Nested Choice Patterns**

- **Jump Tables**

- **Goto statements**
  - **States = program locations**
  - **Transitions = goto**

- **Long jumps**
  - **Setjmp: records the environment**
 - **Longjmp: restores the environment**

- **Object-oriented: State design pattern**

TU/e
Technische Universiteit
Eindhoven
University of Technology
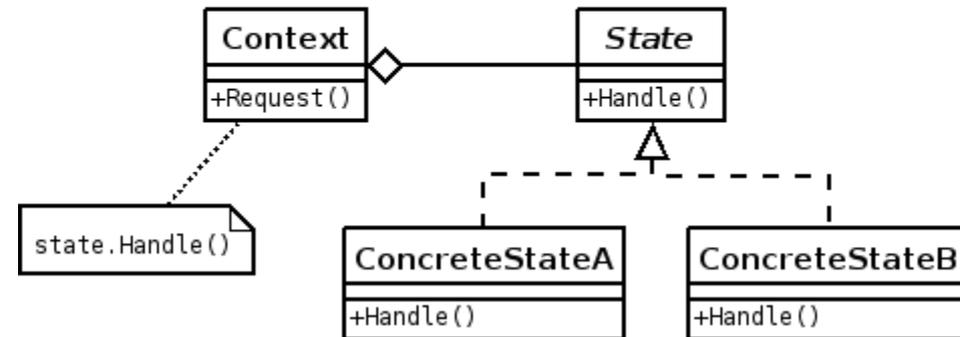
Explicit State Machines: Architecture Reconstr.

- **Recognize a pattern**
- **And create the state machine**

Dennie van Zeeland 2008

ische Universiteit
Eindhoven
University of Technology
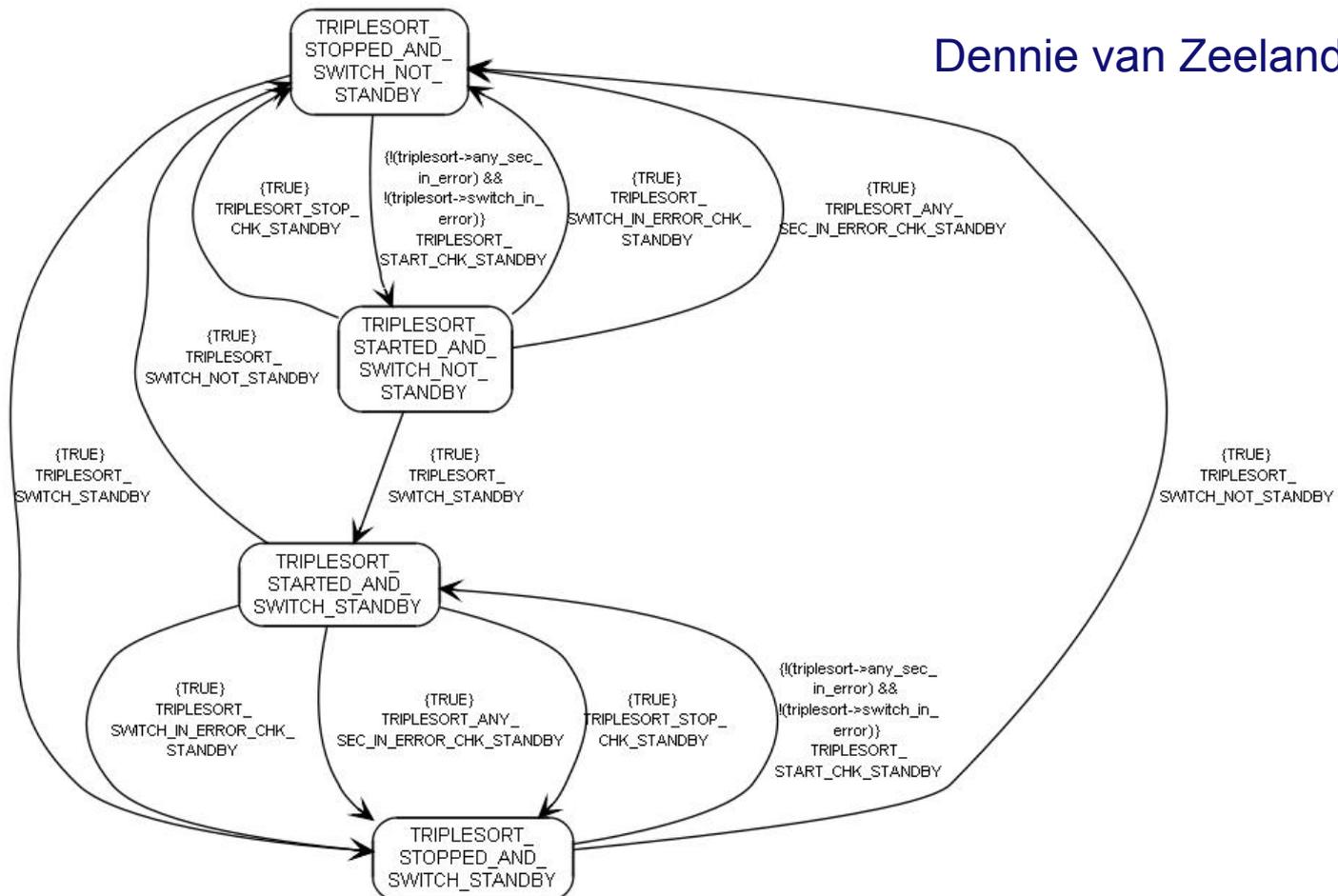
Implicit State Machines

- **States determined by field values**

- **Transitions – modification of the field values by methods**

http://Tapulous.com/staff/

```
class Document {
    …
    Loan loan = null;

    public void addLoan(Loan ln) {
        loan = ln;
    }
    public void removeLoan() {
        loan = null;
    }
}
```

Tonella Potrich

S0 {loans=empty}

addLoan   removeLoan

S1 {loans=one}

addLoan   removeLoan

addLoan   removeLoan

S2 {loans=many}
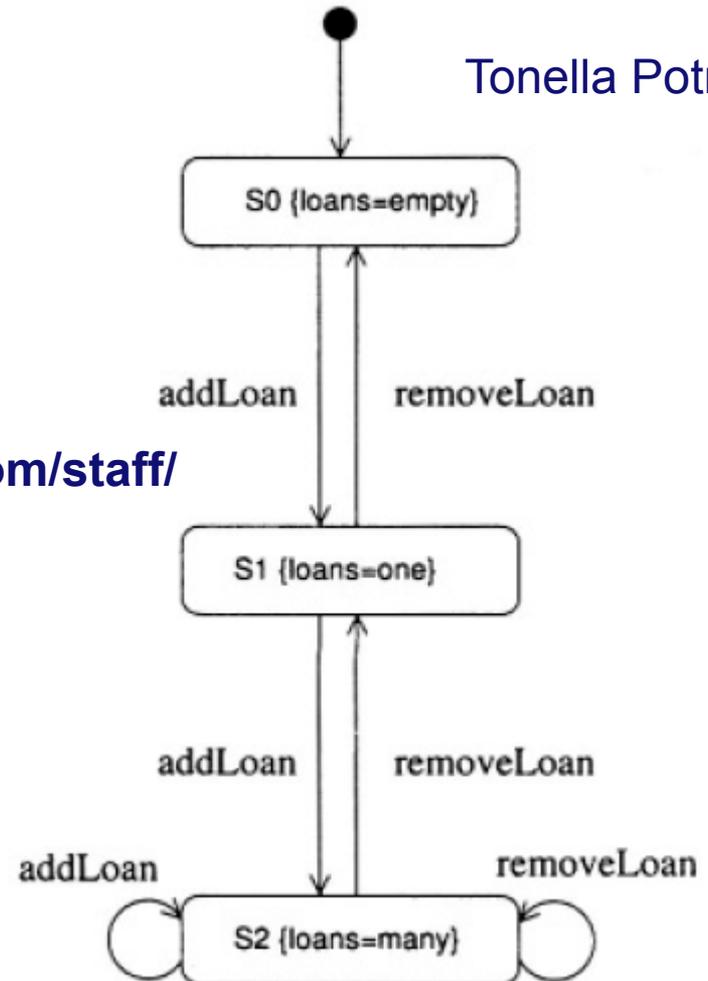
States

- **"States determined by field values"**
  - **Which fields to chose?**

```
class Document {
    int documentCode;
    String title;
    String authors;
    String ISBNCode;
    Loan loan = null;
    static int nextDocumentCodeAvailable = 0;
    …
}
```

Cannot be changed by the class methods (except for constructor)

Changes but class-level!

Technische Universiteit
Eindhoven
University of Technology

States

- **"States determined by field values"**
  - **Which fields to chose?**
  - **Which values to chose?**

```
Loan loan
```

Objects are pointers: many possible values and not all of them are meaningful

- **Often groups (equivalence classes) are more meaningful than individual values:**
  - **"Pointer is null" vs. "pointer is not null"**
  - **Zero, one or more loans**
  - **"temperature is positive and precipitation < 250 mm per year"**

TU/e Technische Universiteit **Eindhoven** University of Technology

Abstract interpretation (Cousot, Cousot)

- **Abstract value – representation of group of concrete values**
  - "positive"   {x | x > 0}
  - "woman"   {x | person(x) /\ female(x) /\ adult(x)}

- **Abstract domain – set of all possible abstract values**
- **Abstraction – mapping from concrete to abstract val:**
  - $\alpha(5)$ = positive

Abstract Interpretation

Requirement
$\langle A, \alpha, C, \gamma \rangle$ should form a Galois connection

Concrete domain — Abstraction function α → Abstract domain

f : Computation is difficult or impossible

Approxi-mation

f_A : Computation is easier

Concrete domain ← Concretization function γ — Abstract domain

TU/e
Technische Universiteit
Eindhoven
University of Technology
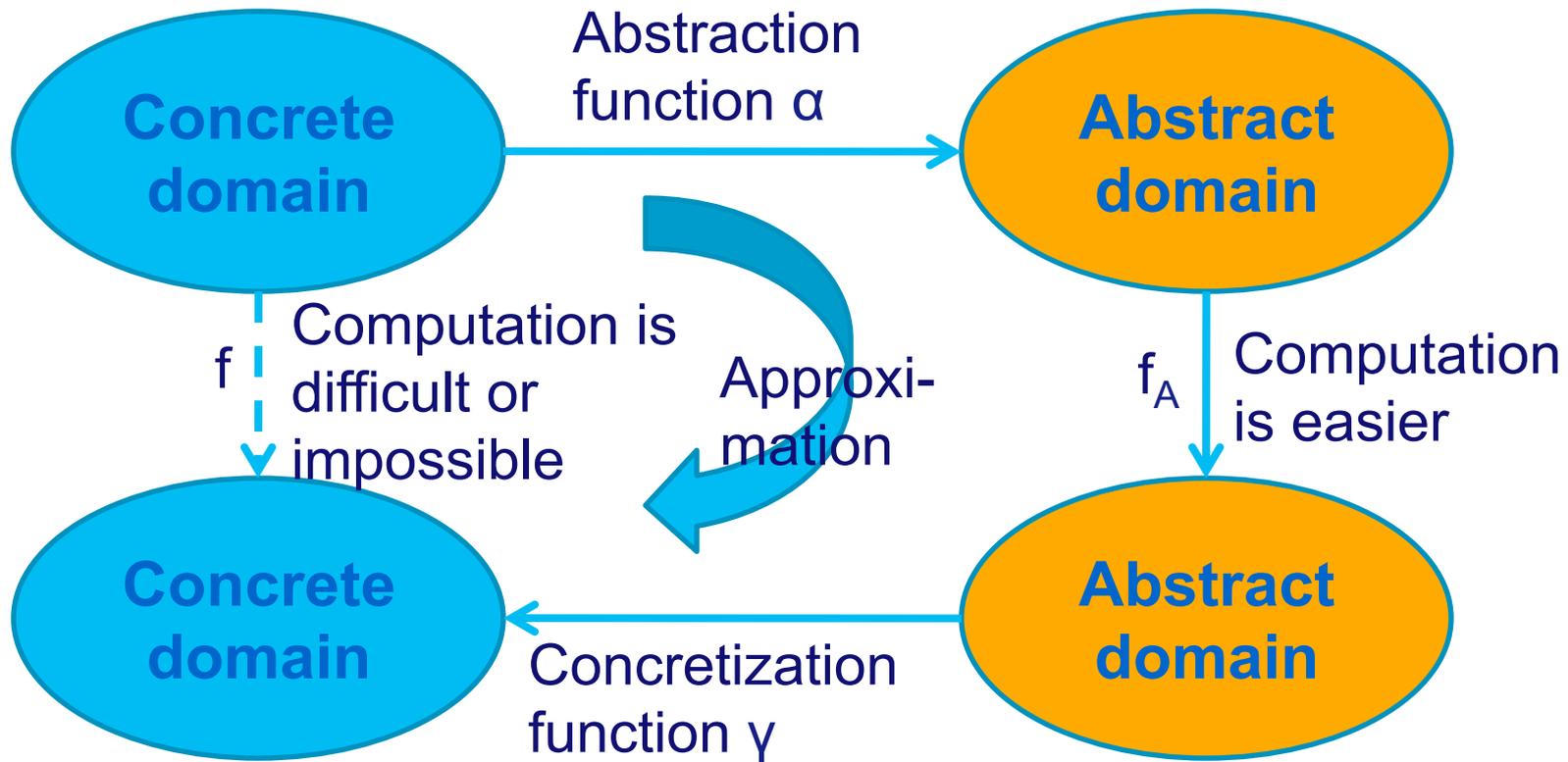
Abstract Interpretation: Example

- $2173 \times 38 = 81574$ or $2173 \times 38 = 82574$?

- **Casting out nines:**
    - **Sum the digits in the multiplicand n_1, multiplier n_2 and the product n to obtain s_1, s_2 and s.**
    - **Divide s_1, s_2 and s by 9 to compute the remainder, that is, $r_1 = s_1 \bmod 9$, $r_2 = s_2 \bmod 9$ and $r = s \bmod 9$.**
    - **If $(r_1 \times r_2) \bmod 9 \neq r$ then multiplication is incorrect**

- **The algorithm returns "incorrect" or "don't know"**

- **What are the concrete/abstract domains? Abstraction/ concretisation functions?**

State machines?

- **Concrete domain:**
  - **values of the class fields**
  - **functions over the concrete domain:**
    - **methods**
    - **constructors must be applied first**
- **Abstract domain:**
  - **representation of the sets of the values of the class fields**
  - **which sets?**
    - **standard techniques (null/not-null, 0-1-many, pos-0-neg)**
    - **well-studied domains (intervals, octagons)**
    - **determined by comparisons in the code**
  - **functions over the abstract domain**

State machines reconstruction

1. **Initialization:**
   - **initStates = $\varnothing$**
   - **s_0 = α(initialization values)**
   - **SM = {<•,'', s_0 >}**

2. **For every constructor cs:**
   - **s = $cs_A(s_0)$**
   - **initStates = initStates U {s}**
   - **SM = SM U {< s_0,cs,s>}**

3. **toGoStates = initStates**

4. **while not empty(toGoStates)**
   - **r = select(toGoStates)**
   - **toGoStates = toGoStates \ {r}**
   - **for each method m**
     - **s = $m_A(r)$**
     - **SM = SM U {<r,m,s>}**
     - **toGoStates = toGoStates U {s}**

Example

```
class Document {
    …
    Loan loan = null;

    public void addLoan(Loan ln) {
        loan = ln;
    }
    public void removeLoan() {
        loan = null;
    }
}
```

Why?

removeLoan

addLoan

s_0 (loan = null)

addLoan

removeLoan

addLoan
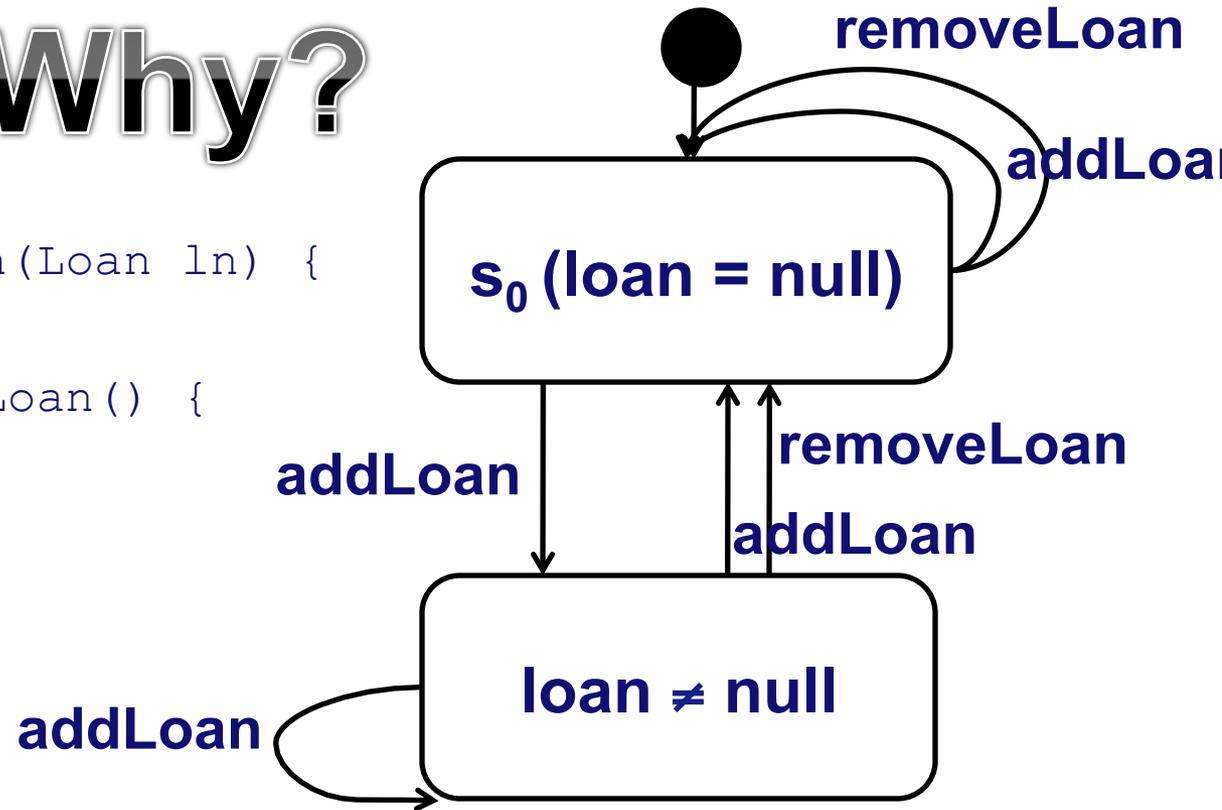
loan ≠ null

addLoan

Hidden assumptions:
- **precondition addLoan:** ln ≠ null
- **precondition removeLoan:** loan ≠ null

State machine reconstruction: Summary

- **State machines**
  - **Popular for object behaviour specification**
  - **Can be used for model checking**
- **Explicit state machines**
  - **Designed by a developer**
  - **Patterns**
- **Implicit state machines**
  - **Abstract interpretation**

Part 2: Architecture

- **As intended**

evolution

- **As described**
  - **Architecture Description Languages**
    - **Should express different views**

evolution

- **As implemented**
  - **Code, deployment**
  - **From code to architecture: reverse engineering**
    - **Should extract different views**

Current ADLs target

- **Correctness: e.g., Wright [Allen 1997]**
  - based on Communicating Sequential Processes
  - defines consistency properties
  - that can be checked using model checking

- **System config. and code generation, e.g.,**
  - **Fractal [Bruneton, Coupaye, Leclercq, Quema, Stefani 2004]**
    - components: primitive (code) and composite (group)
    - code generation
  - **ArchJava [Aldrich, Chambers, Notkin 2002]**
    - extension of Java
      - components, ports, connections
    - enforces communication integrity

Technische Universiteit
Eindhoven
University of Technology

Current ADLs target

- **Correctness: e.g., Wright [Allen 1997]**

```
public component class Parser {
    public port in {
        provides void setInfo(Token symbol,
                                SymTabEntry e);
        requires Token nextToken()
        throws ScanException;
    }
    public port out {
        provides SymTabEntry getInfo(Token t);
        requires void compile(AST ast);
    }
    …
```

- **Sy...**

  - ...

    - ...p)

    - ...

  - **A...**

    - …

        - **components, ports, connections**

   - **enforces communication integrity**

Major problems with current ADLs

- **No support for evolution during the execution**
  - **Architecture can change during the execution:**
    - client connects to a different server
    - check Müller, Villegas on Runtime evolution in
  - **Snapshots easily become obsolete**

- **No support for evolution due to change in requirements**
  - **Wright**
    - model checking: no incremental approach
    - minor modification $\Rightarrow$ everything should be rechecked
  - **ArchJava**
    - no real separation of architecture/implementation
    - overtly complex code
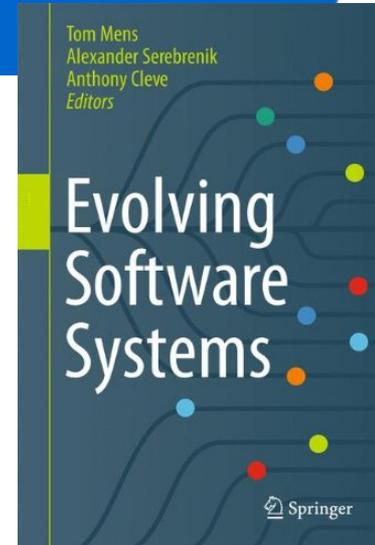
First attempts at solutions: Extended Wright
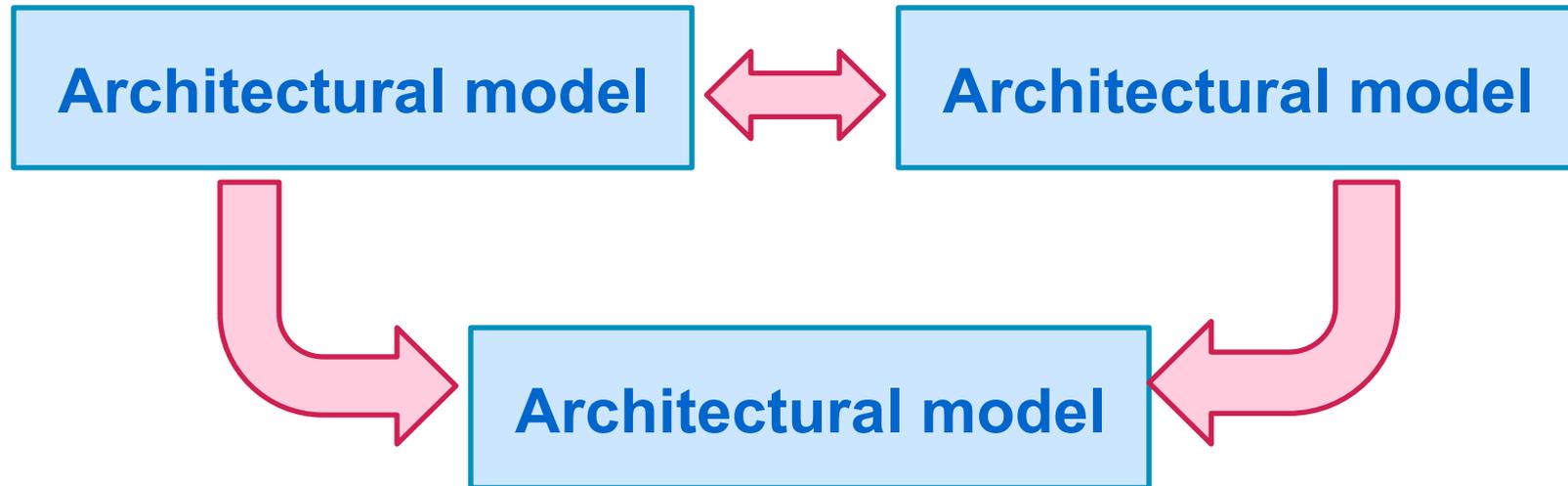
- **Event-driven reconfiguration**
  - **Component description should be adapted to describe when/which reconfigurations are permitted**
  - **Incomplete separation of the reconfiguration policies**
- **Can be used to model evolution, but**
  - **Finite number of models**

TU/e
Technische Universiteit
Eindhoven
University of Technology

Architecture Description Languages

- **Minimal support for evolution**
  - **Only run-time**

- **Use of an ADL can even hinder evolution**

- **On-going research**

TU/e
Technische Universiteit
Eindhoven
University of Technology

Conclusions

- **Behaviour reconstruction**
  - **Classes or objects**
  - **Sequence diagrams or state machines**
  - **Statically or dynamically**
  - **Visualization**

- **ADL**

Technische Universiteit
Eindhoven
University of Technology