

Architecture Evolution

Alexander Serebrenik



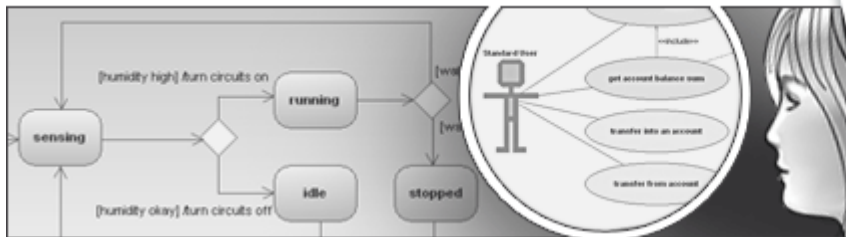
TU / **e**

Technische Universiteit
Eindhoven
University of Technology

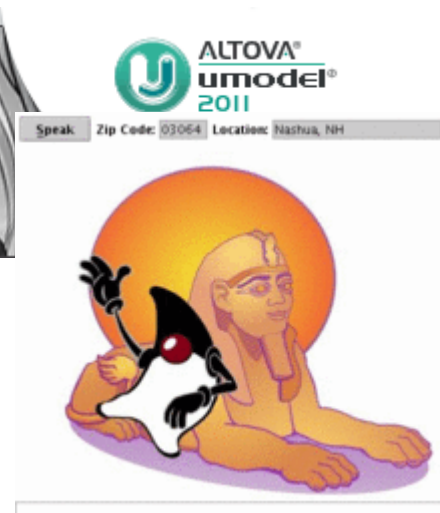
Where innovation starts

Assignments: Reminder

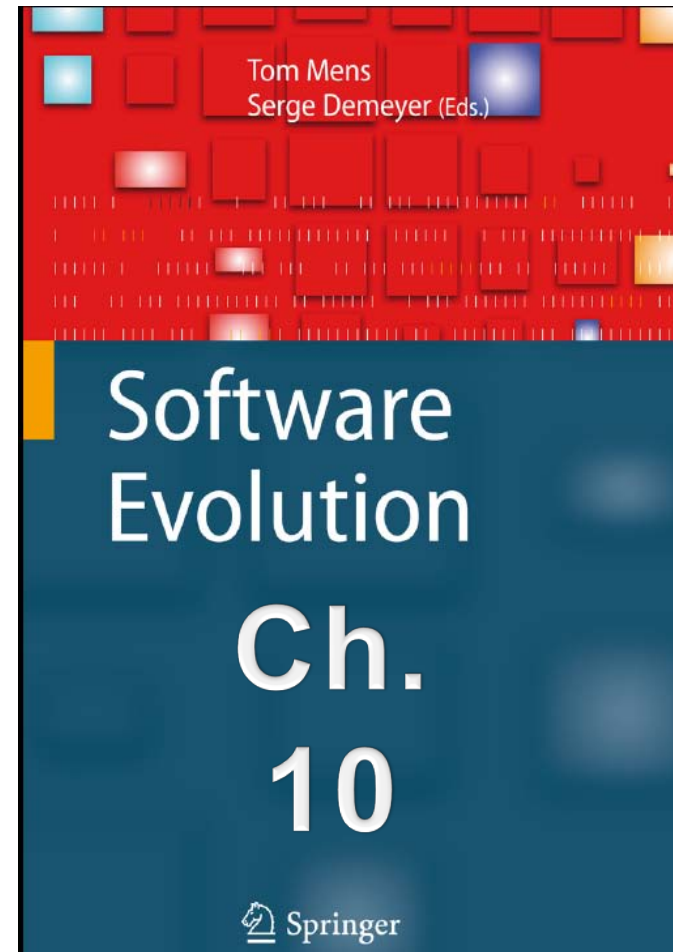
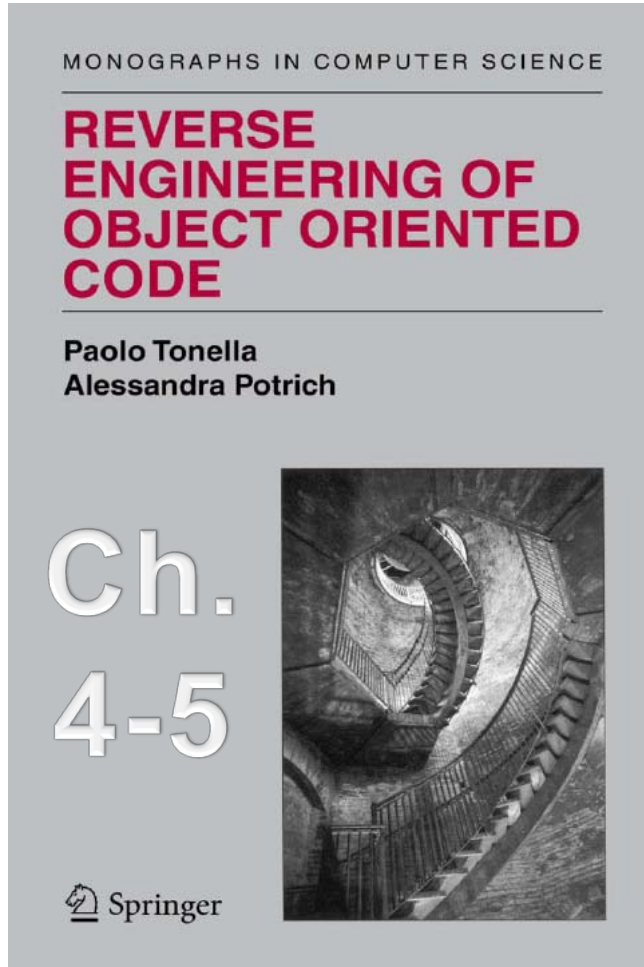
- **Assignment 2:**
 - **Deadline: Today!**
 - **Individual**
- **Assignment 3:**
 - **Deadline: March 16, 23:59**
 - **Individual or pair**
 - **Architecture reconstruction with**



Sphinx 4.1 beta release 2 and 4



Sources



Part 1: Where are we now?

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

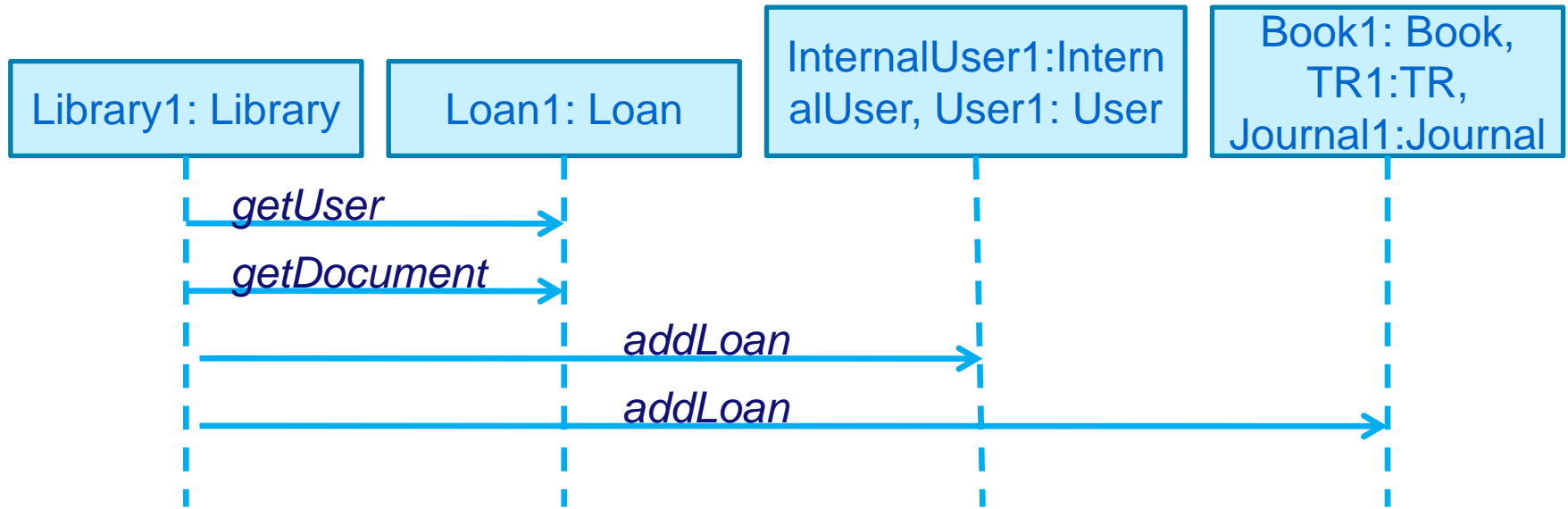
- **This week: behaviour**
 - **Sequence diagrams**
 - **State diagrams (State machines)**

Static vs. Dynamic Analysis

- **Dynamic (execution)**
 - + More precise
 - 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 week	Today
	Dynamic		

Static analysis of behavioural models



- **Columns**
 - Containers
 - Objects vs. classes
- **Method invocations**
 - Resolving the calls
 - Invocation order

Recapitulation: Object Flow Graph

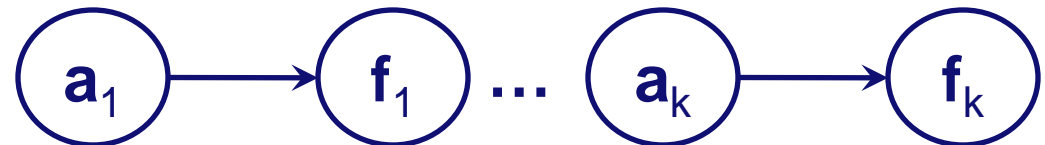
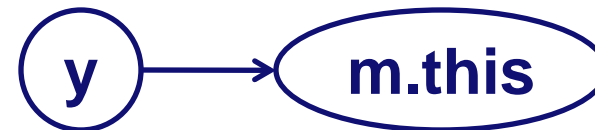
- Vertices: variables, fields, method parameters

- $x = y$



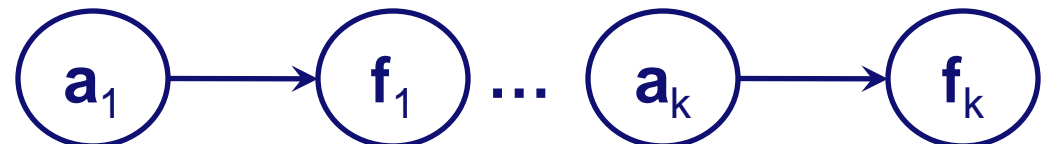
- $x = y.m(a_1, \dots, a_k)$

- Method $m(f_1, \dots, f_k)$



- $x = \text{new } c(a_1, \dots, a_k)$

- cs – constructor of c

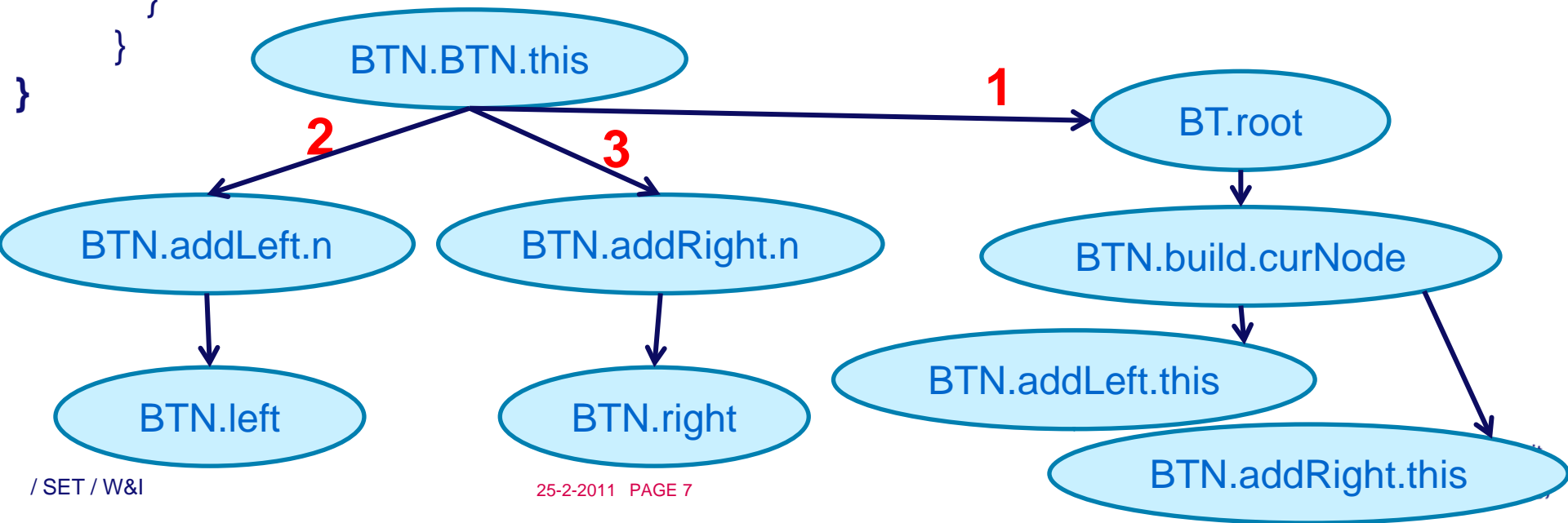


Objects vs. Classes

```
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());  
  }  
}
```

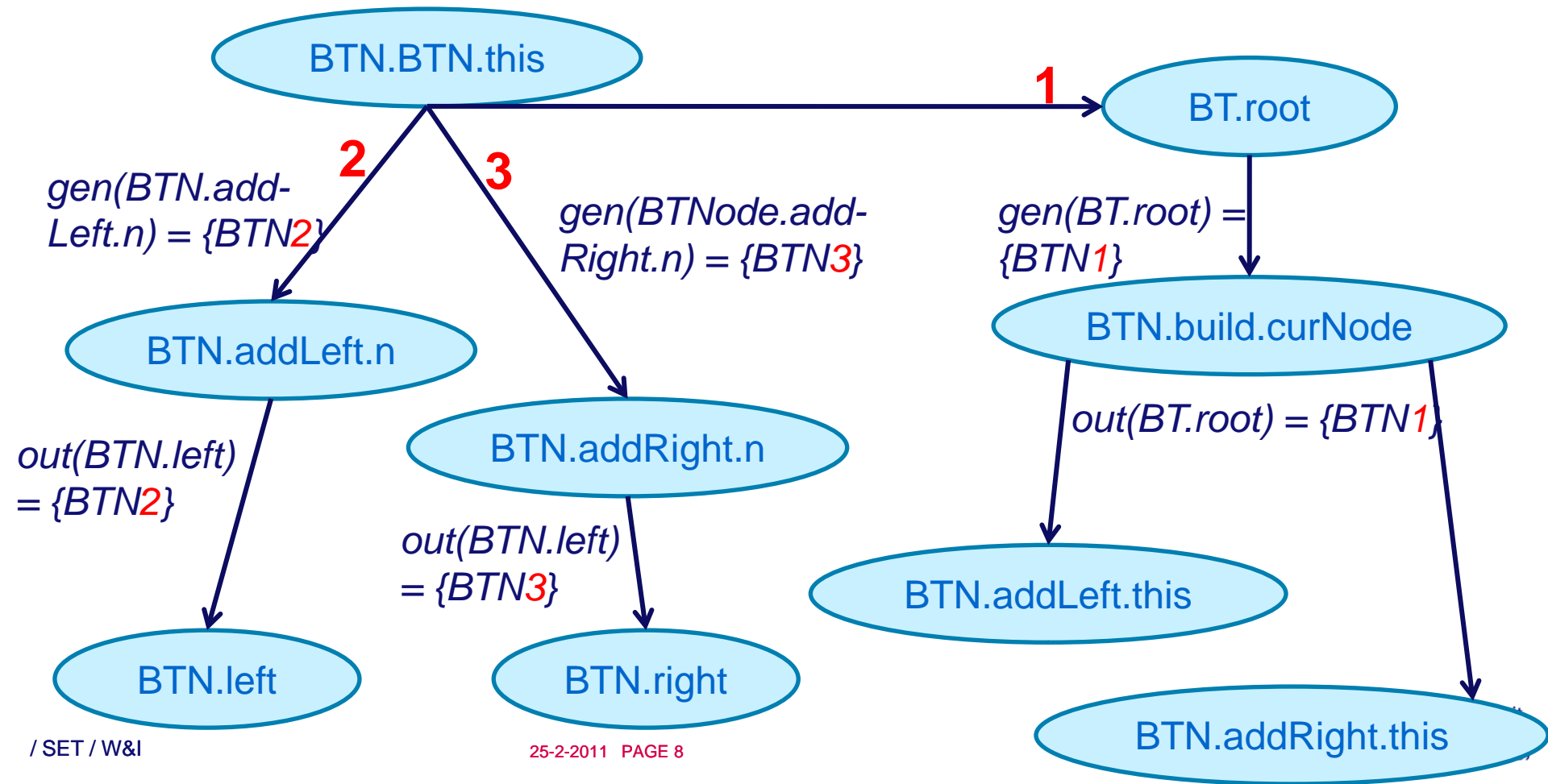
```
class BTNode {  
  BTNode left, right;  
  public void addLeft(BTNode n) {  
    left = n;  
  }  
  public void addRight(BTNode n) {  
    right = n;  
  }  
}
```

Tonella, Potrich 2005

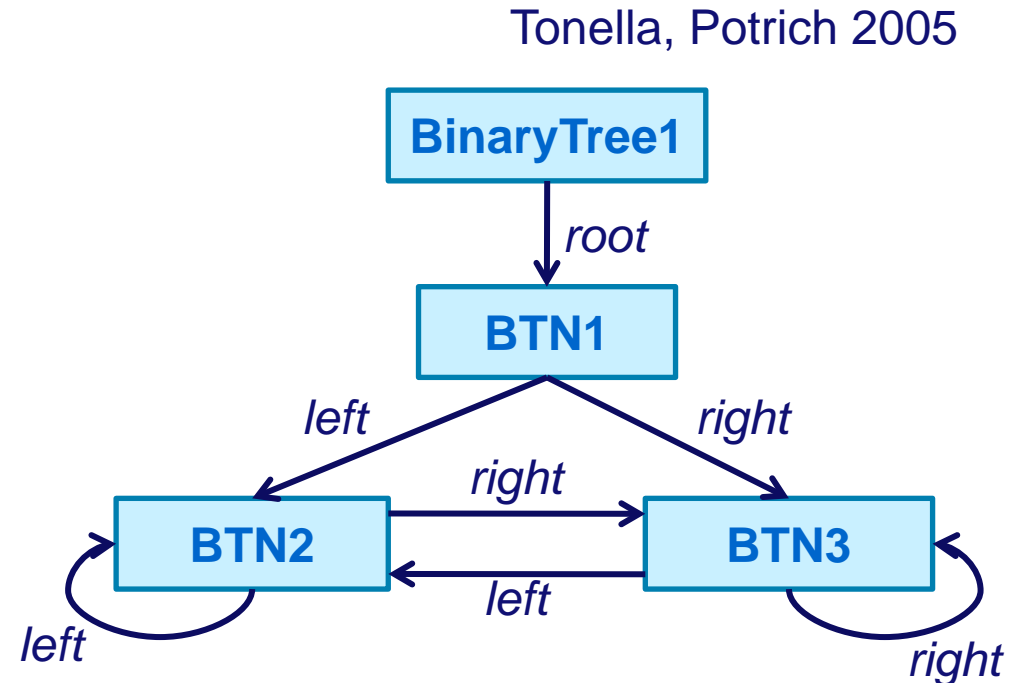
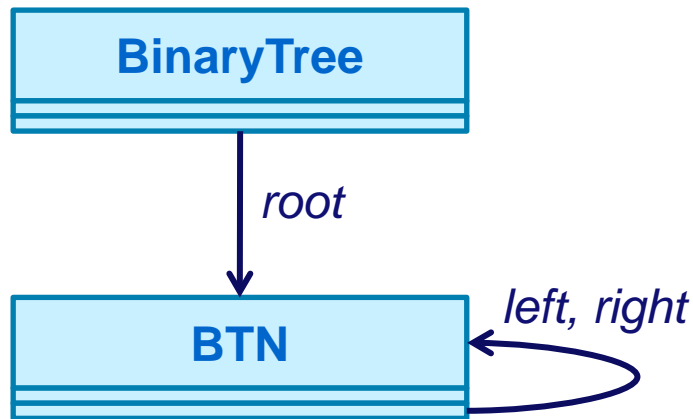


Solution

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



Objects vs. Classes



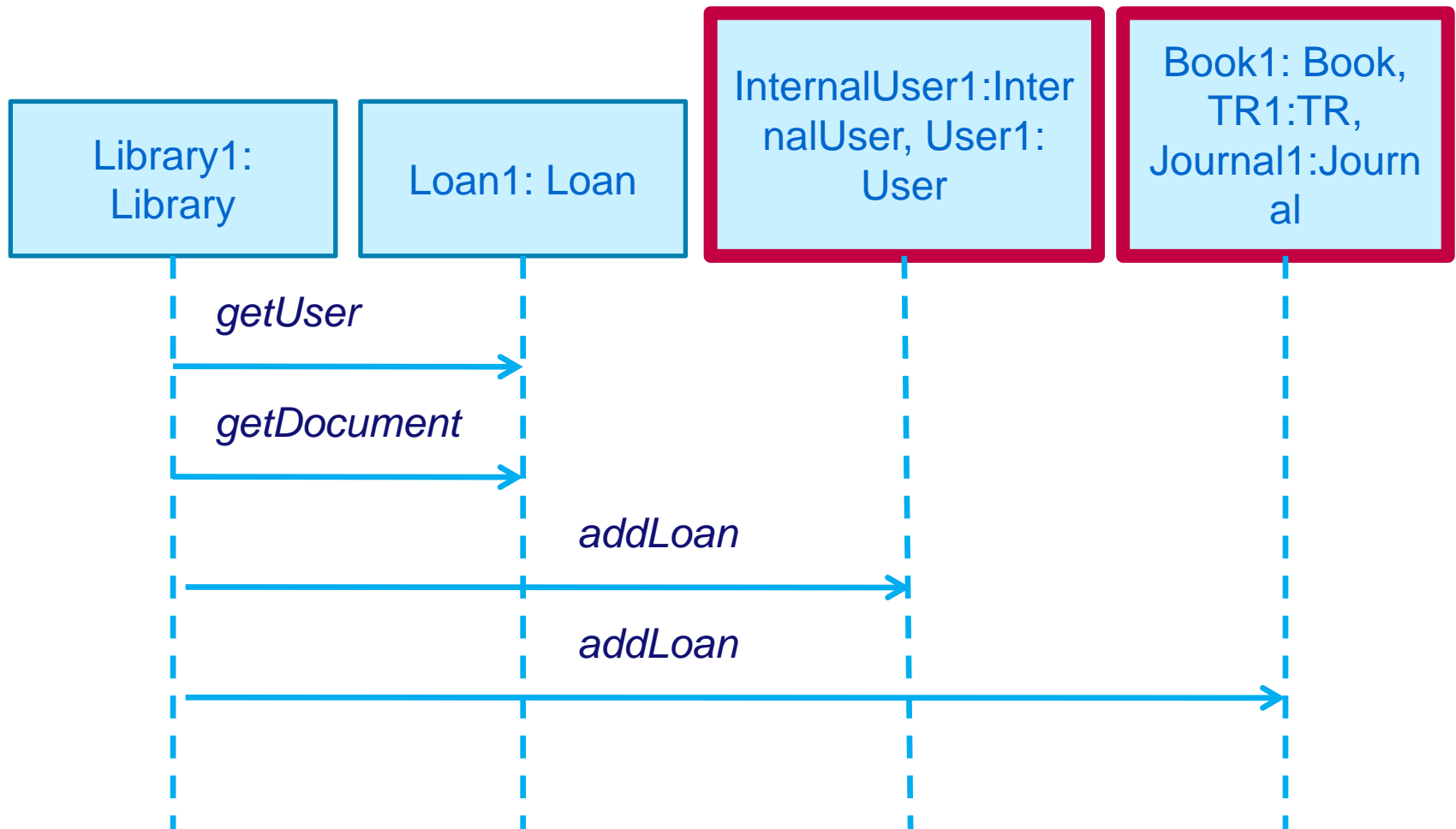
- We can use the same technique for sequence diagrams!

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, not objects! And containers...
 - 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?**
 - 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 intertwined laws
- Spec is inconsistent with *GlassFish*

Code Example: Enterprise Java Bean

```
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 {...}
```

```
}
```

Example (cont.): XML deployment descriptor

```
<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>
```


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

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:

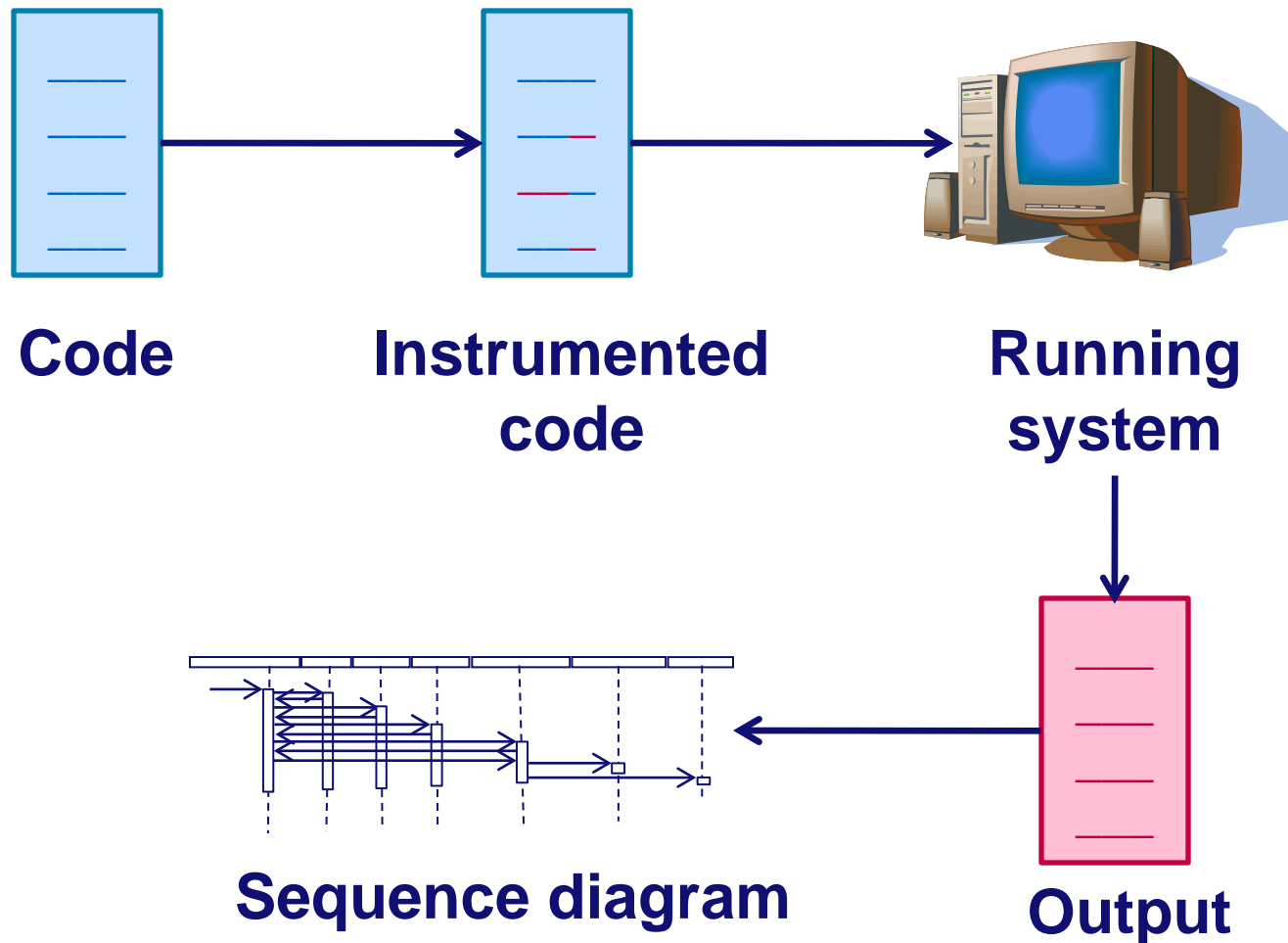
```
for (i = 0; i < arrayOfInts.length; i++) {  
    for (j = 0; j < arrayOfInts[i].length; j++) {  
        if (arrayOfInts[i][j] == searchfor) {  
            foundIt = true;  
            break search;  
        }  
    }  
}
```

Rountev, Volgin, Reddoch
PASTE'05

Static analysis so far...

- **Architecture reconstructor has to decide**
 - **Classes or objects**
- **Methods**
 - **Resolve the method calls**
 - **Invocation order**
 - **May be complex (interceptors)**

Dynamic analysis



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**

How to instrument the code?

- **Classes:** augment with the object identifier

- Java: hashCode, rmi.ObjID, etc...

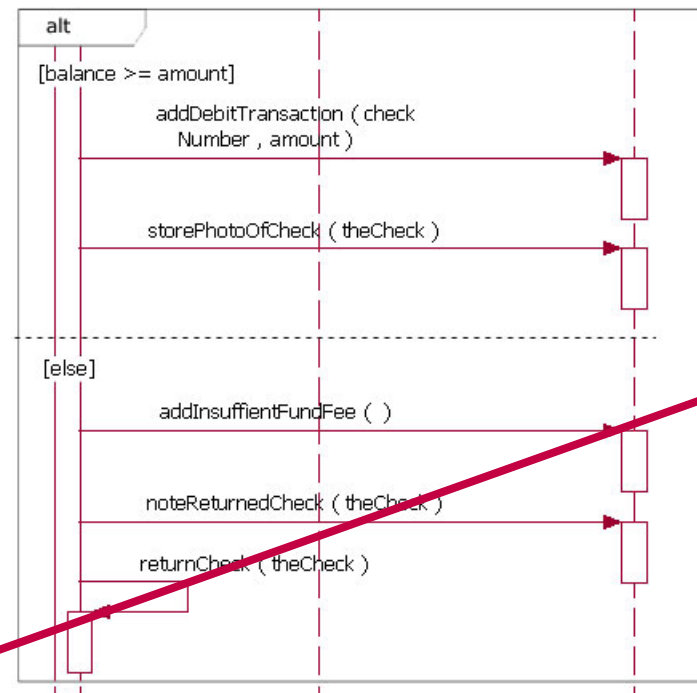
- Add to the trace

<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

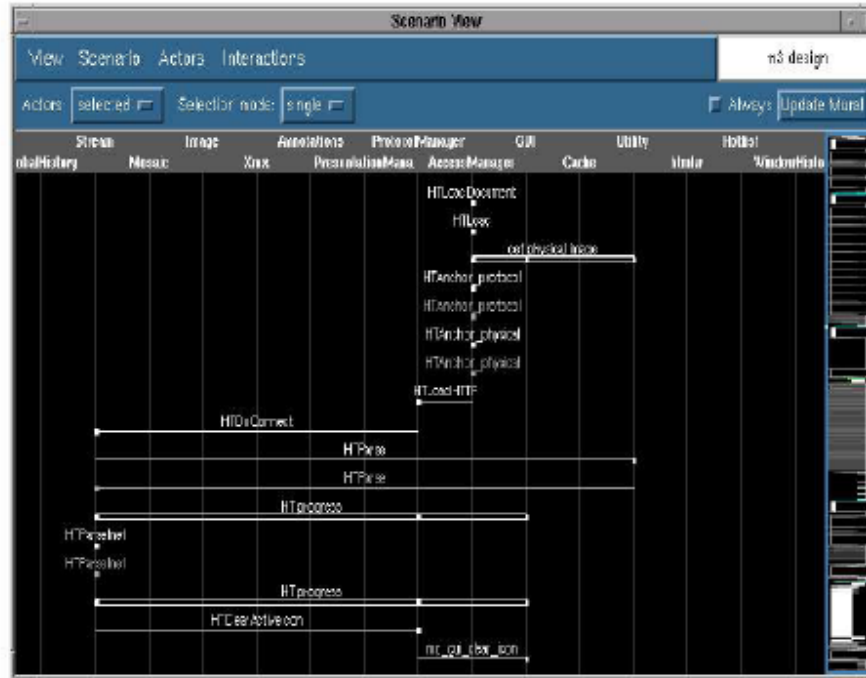
- One process in a printer component
 - 129 calls from the main function
 - 18 objects (1 per class) involved
 - Picture does not fit the slide
- Solutions
 - Smaller scale
 - “interesting” top-level functions
 - “interesting” calls in these functions
 - “interesting” target objects/classes
 - Different visualization

Require a priori knowledge

Better visualization (1)

Jerding, Stasko, Ball 1997

Message-
flow diagram

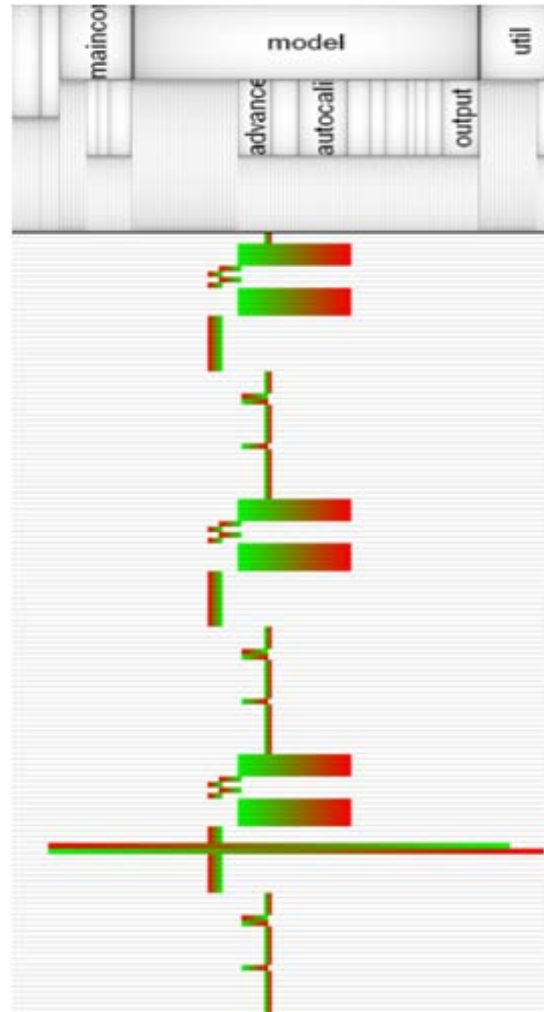


Information
mural view:
general
overview in
small

One can define **interaction patterns** and search for their appearances.

Better visualization (2)

Holten, 2009



Hierarchy of packages and classes

Arrow:
from **green**
to **red**

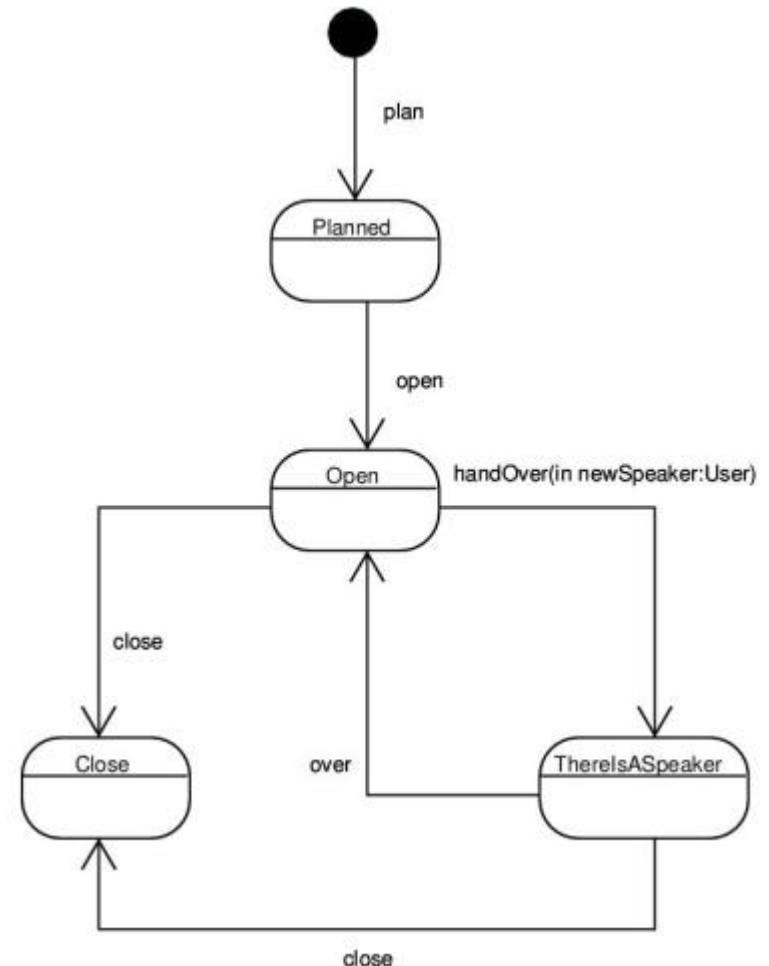
Repeated pattern and exceptional behaviour are clearly visible. Solves the blurring up problem by special visualization technique

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>

Explicit State Machine Patterns

- **Nested Choice Patterns**

- **Switch/Case – Switch/Case**
- **Switch/Case – If/Else**
- **If/Else – Switch/Case**
- **If/Else – If/Else**

```
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;
    ...
}
```

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**

```
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);
    }
}
```

Explicit State Machine Patterns

- **Nested Choice Patterns**

- **Jump Tables**

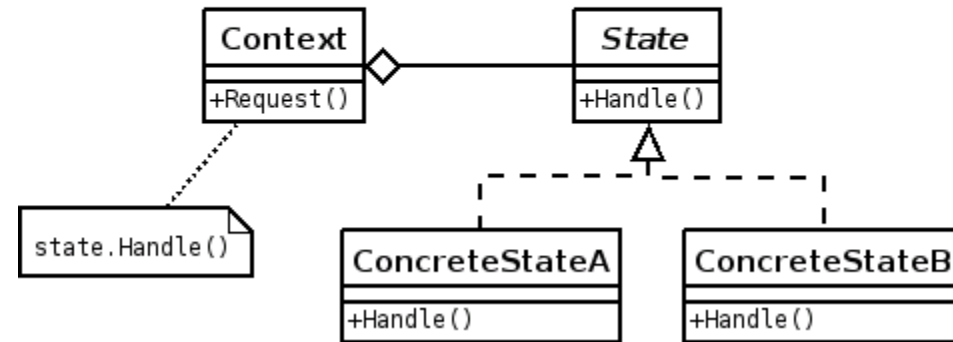
- **Goto statements**

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

- **Long jumps**

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

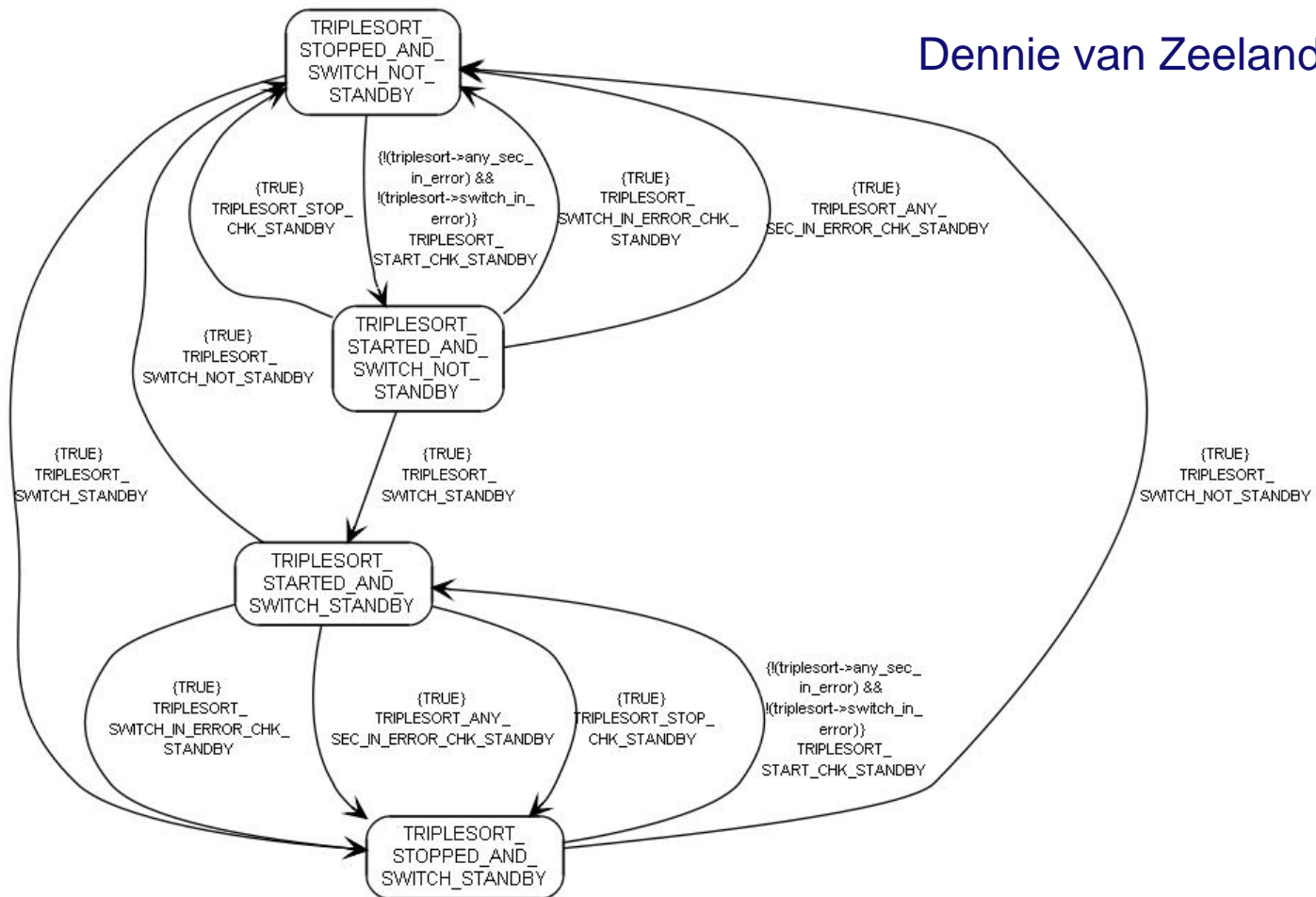
- **Object-oriented: State design pattern**



Explicit State Machines: Architecture Reconstr.

- Recognize a pattern
- And create the state machine

Dennie van Zeeland 2008

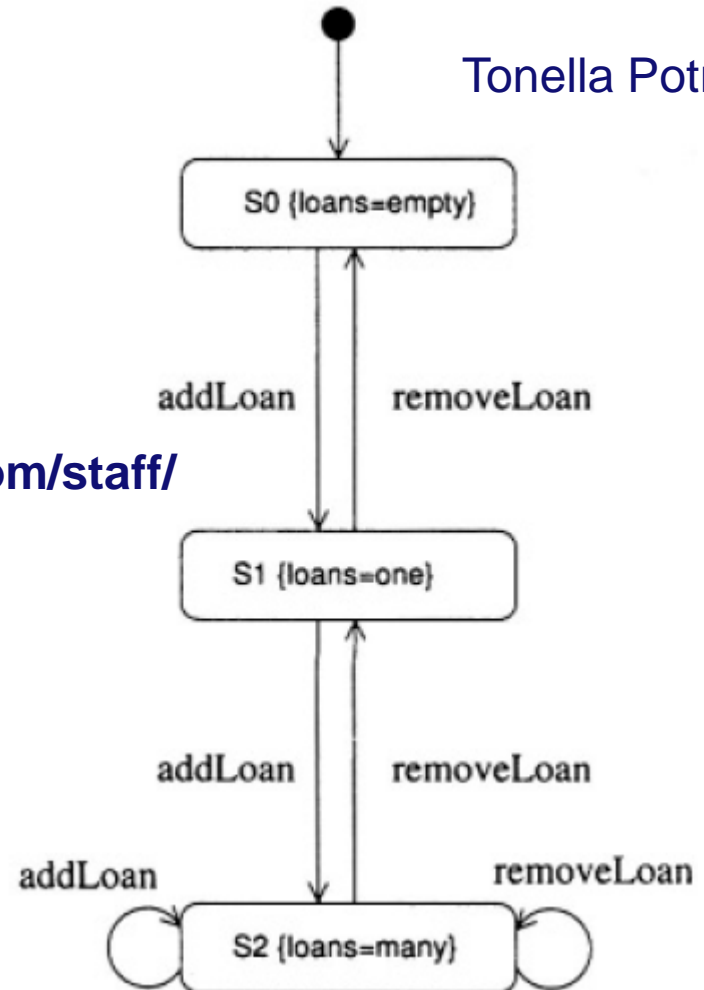


Implicit State Machines

- 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;  
    }  
}
```



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!

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

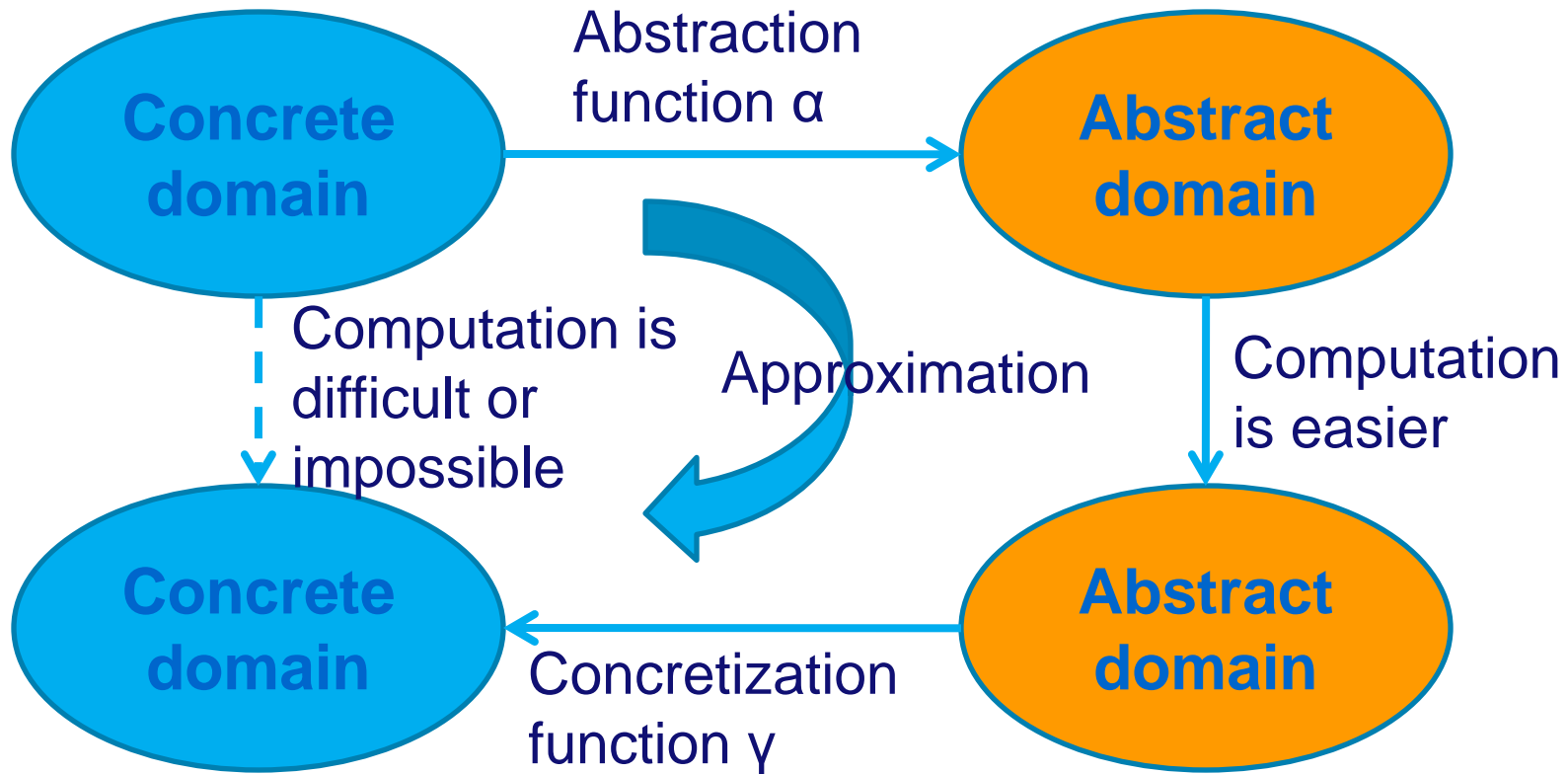
- Often **groups (equivalence classes)** are more meaningful than individual values:
 - “Pointer is null” vs. “pointer is not null”
 - Zero, one or more loans
 - “temperature is positive and precipitation < 250 mm per year”

Abstract interpretation (Cousot, Cousot)

- **Abstract value** – representation of group of concrete values
 - “positive” represents $\{x \mid x > 0\}$
 - “woman” represents $\{x \mid \text{female}(x) \wedge \text{adult}(x)\}$
- **Abstract domain** – set of all possible abstract values
 - Complete semi-lattice
 - Partially ordered (\leq)
 - For each two elements there exists a unique **lub**
- **Abstraction** – mapping from concrete to abstract val:
 - $\alpha(5) = \text{positive}$

Abstract Interpretation

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

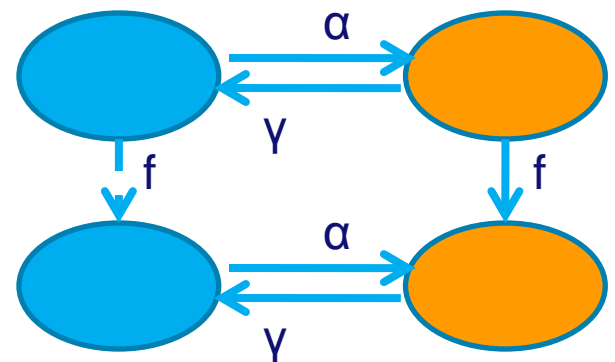


Abstract Interpretation: Example

- $2173 \times 38 = 81574$ 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?

To ensure correctness of the abstraction

- $\langle \mathbf{A}, \alpha, \mathbf{C}, \gamma \rangle$ should be a **Galois connection**
 - $\langle \mathbf{A}, \leq_{\mathbf{A}} \rangle$ and $\langle \mathbf{C}, \leq_{\mathbf{C}} \rangle$ are complete semi-lattices
 - Recap: \leq and a unique **lub**
 - $\alpha: \mathbf{C} \rightarrow \mathbf{A}$ and $\gamma: \mathbf{A} \rightarrow \mathbf{C}$ are monotonic
 - $\gamma \circ \alpha: \mathbf{A} \rightarrow \mathbf{A}$ and $\alpha \circ \gamma: \mathbf{C} \rightarrow \mathbf{C}$ satisfy
 - $c \leq_{\mathbf{C}} (\gamma \circ \alpha)(c)$ for all $c \in \mathbf{C}$
 - $(\alpha \circ \gamma)(a) \leq_{\mathbf{A}} a$ for all $a \in \mathbf{A}$
- **Galois connection guarantees**
 - $(\alpha \circ f_{\mathbf{C}} \circ \gamma)(a) = a$
 - $f_{\mathbf{C}}(c) \leq_{\mathbf{C}} (\gamma \circ f_{\mathbf{A}} \circ \alpha)(c)$ (*)



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
 - define to satisfy (*)

State machines reconstruction

1. Initialization:

- $\text{initStates} = \emptyset$
- $s_0 = \alpha(\text{initialization values})$
- $\text{SM} = \{ \langle \bullet, \text{''}, s_0 \rangle \}$

2. For every constructor cs :

- $s = \text{cs}_A(s_0)$
- $\text{initStates} = \text{initStates} \cup \{s\}$
- $\text{SM} = \text{SM} \cup \{ \langle s_0, \text{cs}, s \rangle \}$

3. $\text{toGoStates} = \text{initStates}$

4. while not

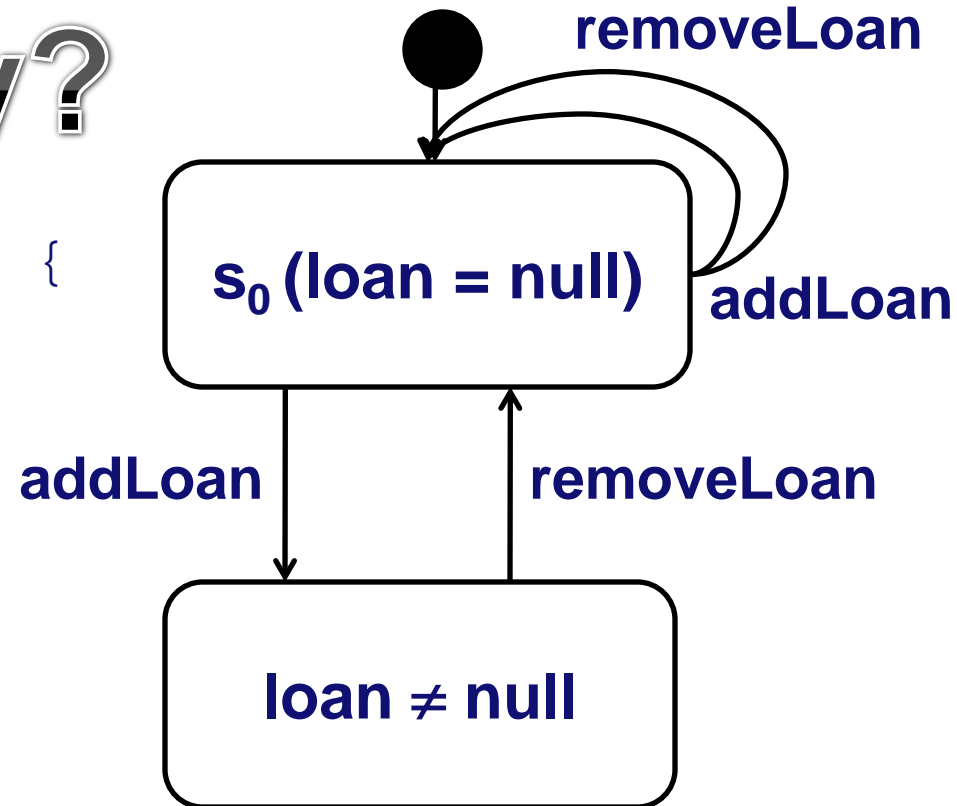
$\text{empty}(\text{toGoStates})$

- $r = \text{select}(\text{toGoStates})$
- $\text{toGoStates} = \text{toGoStates} \setminus \{r\}$
- for each method m
 - $s = m_A(r)$
 - $\text{SM} = \text{SM} \cup \{ \langle r, m, s \rangle \}$
 - $\text{toGoStates} = \text{toGoStates} \cup \{s\}$

Example

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

Why?



Hidden assumptions:

- precondition addLoan: $ln \neq null$
- precondition removeLoan: loan $\neq null$

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 (last week)

- **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]**

```
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);  
  }  
  ...  
}
```

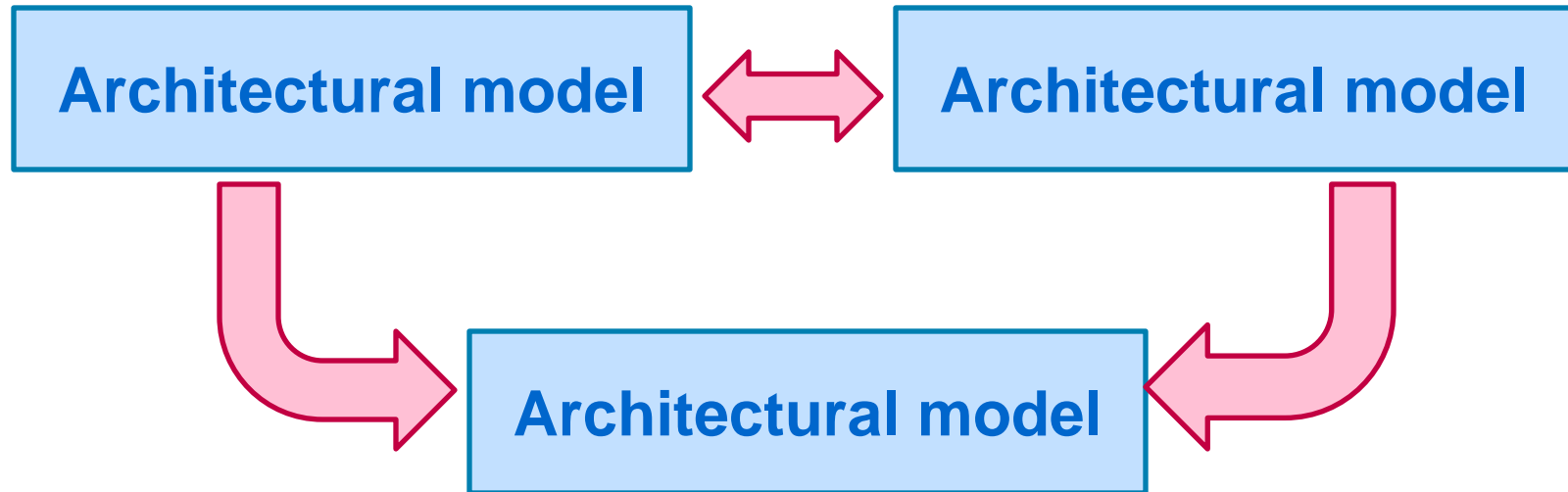
p)

- 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**
 - **Snapshots easily become obsolete**
- **No support for evolution due to change in requirements**
 - **Wright**
 - **model checking: no incremental approach**
 - **minor property/model 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

Architecture Description Languages

- **Minimal support for evolution**
 - **Only run-time**
- **Use of an ADL can even hinder evolution**
- **On-going research**

Conclusions

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

- **ADL**

- **Next week (March 2): invited lecture by Stijn Hoop**
- **March 9: Carnaval**
- **March 16: Deadline Assignment 3**