Assignment 3: Reminder

- Deadline: Next Monday, 23:59
- Individual or pair
- Questions?

Part 1: Where are we now?

- Last week: architecture
  - Structure (class and package diagrams)
  - Where have we been in the table?

- 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

<table>
<thead>
<tr>
<th>Analysis technique</th>
<th>Object of the analysis</th>
<th>Structure</th>
<th>Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>Last week</td>
<td>Today</td>
<td></td>
</tr>
<tr>
<td>Dynamic</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Static analysis of behavioural models

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

```java
class BT {
    BTNode root;
    public void build() {
        root = new BTNode();
        BTNode curNode = root;
        while (…) {
            curNode.addLeft(new BTNode());
            curNode.addRight(new BTNode());
        }
    }
}
```

Objects vs. Classes

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

Next step: resolving method calls

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

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

Solution

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

```
1 2 3
```

```
• Generate
• Increment the counter for every “new BTNode”
• Propagate this information through the OFG
```

```
1 2 3
```

```
• We can use the same technique for sequence diagrams!
```

```
• We can use the same technique for sequence diagrams!
```

```
• Intuitively, C.m() ⇒⇒⇒ D
• But
  • If
  • O
• NB: out are sets (polymorphism)
```

```
• Intuitively, C.m() ⇒⇒⇒ D
• But
  • If
  • O
• NB: out are sets (polymorphism)
```

```
Logging, security, performance
```

```java
public class importantClass {
    public String importantMethod() {
        …
    }
}
```

```
Logging, security, performance
```

```java
public class importantClass {
    public String importantMethod() {
        …
    }
}
```

Interceptor

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

```
• How?
  • XML file
  • Annotation
• At what level?
  • Bean class
  • Method
• Where?
  • Bean class
  • Superclass
  • Separate class
  • Injected bean
```

```
• 9 complex and interwined laws
• Spec is inconsistent with GlassFish
```

```
• 9 complex and interwined laws
• Spec is inconsistent with GlassFish
```
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 {...}
}
```

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>
```

Reverse engineered sequence diagram

```
 Sounds interesting?
 Talk to me!
 Opportunity for a Capita Selecta assignment.
```

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

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

Better visualization (1)

Jerding, Stasko, Ball 1997

Information mural view: general overview in small

Message-flow diagram

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.

Explicit State Machine Patterns

• Nested Choice Patterns
  • Switch/Case – Switch/Case
  • Switch/Case – If/Else
  • If/Else – Switch/Case
  • If/Else – If/Else

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

Explicit State Machine Patterns

• Nested Choice Patterns
  • Jump Tables
    • Goto statements
      • States = program locations
      • Transitions = goto
  • Object-oriented: State design pattern
    • Long jumps
      • Setjmp: records the environment
      • Longjmp: restores the environment
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/

Dennie van Zeeland 2008

States

- “States determined by field values”
- Which fields to chose?
  class Document {
  int documentCode;
  String title;
  String authors;
  Loan loan = null;
  static int nextDocumentCodeAvailable = 0;
  ...
  }

  Cannot be changed by the class methods (except for constructor)
  Changes but class-level!

Abstract interpretation (Cousot, Cousot)

- Abstract value – representation of group of concrete values
  - “positive” represents \{x | x > 0\}
  - “woman” represents \{x | female(x) ∧ 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:
    - \(α(5) = \text{positive}\)

Abstract Interpretation

\(\langle A, α, C, γ \rangle\) should form a Galois connection

Computation is easier

Concrete domain

Concrete domain

Abstract domain

Abstract domain

Approximation

Concretization function γ

Abstraction function α

\(\leq\)

\(\text{positive}\)
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, $r_1 = s_1 \mod 9$, $r_2 = s_2 \mod 9$ and $r = s \mod 9$.
  - If $(r_1 \times r_2) \mod 9 \neq r$ 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

- $(A, \alpha, C, \gamma)$ should be a Galois connection:
  - $(A, \leq_A)$ and $(C, \leq_C)$ are complete semi-lattices
  - $\alpha: C \rightarrow A$ and $\gamma: A \rightarrow C$ are monotonic
  - $\gamma \circ \alpha$ is Galois connection guarantees:
    - $(\alpha \circ f \gamma \circ f \alpha)(c) = a$
    - $f \gamma f \alpha (c) \leq_A (\gamma \circ f \alpha)(c)$ (*)

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-negative)
    - well-studied domains (intervals, octagons)
    - determined by comparisons in the code
  - functions over the abstract domain:
    - define to satisfy (*)

State machines reconstruction

1. Initialization:
   - initStates = $\emptyset$
   - $s_0 = \alpha(\text{initialization values})$
   - SM = $(\alpha, s_0)$
2. For every constructor $c_s$:
   - $s = c_s A(s_0)$
   - initStates = initStates U {s}
   - SM = SM U $(s, c_s, s)$
3. toGoStates = initStates
4. while not empty(toGoStates)
   - $r = \text{select(toGoStates)}$
   - toGoStates = toGoStates \ {r}
   - for each method $m$
     - $s = m A(r)$
     - toGoStates = toGoStates U {s}

Example

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

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 (last week)

- As intended
- As described
  - Architecture Description Languages
  - Should express different views
- As implemented
  - Code, deployment
  - From code to architecture: reverse engineering
    - Should extract different views

Current ADLs target

- Correctness: e.g., Wright [Allen 1997]
  ```java
  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);
    }
  }
  ```
  - Components, ports, connections
  - Ensures 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 ⇒ 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