UML-Based Testing
Using Sequence Diagrams, Statecharts and OCL Constraints

Dehla Sokenou
Technische Universität Berlin

Contents

• Motivation
• UML-based testing
  • Test case generation
  • Test oracle
• Aspects used for testing
• Summary and outlook
Motivation

- Open issue: testing object-oriented systems
- UML widely used for modelling and specifying
  - Most UML-based testing approaches concentrate just on one diagram type
    - Mainly dynamic diagrams
      - Our idea: combining several diagram types for testing
- Test code integration often expensive
  - e.g. version control
    - Our idea: using dynamic aspects for testing

<table>
<thead>
<tr>
<th>view</th>
<th>diagram type</th>
<th>test use</th>
</tr>
</thead>
<tbody>
<tr>
<td>static</td>
<td>class diagram</td>
<td>preparation, configuration, oracle, test data</td>
</tr>
<tr>
<td></td>
<td>object diagram</td>
<td>configuration</td>
</tr>
<tr>
<td></td>
<td>management diagrams</td>
<td></td>
</tr>
<tr>
<td>dynamic</td>
<td>interaction diagrams</td>
<td>test cases</td>
</tr>
<tr>
<td></td>
<td>activity diagram</td>
<td></td>
</tr>
<tr>
<td></td>
<td>state diagram</td>
<td>oracle, test cases, test data</td>
</tr>
<tr>
<td></td>
<td>use case diagram</td>
<td>coverage</td>
</tr>
<tr>
<td>static</td>
<td>OCL</td>
<td>oracle, test data</td>
</tr>
</tbody>
</table>
UML-Based Testing

- Diagram types have different test relevance
  - Dynamic diagrams and OCL good foundation for testing
- Our approach combines
  - Interaction diagrams and statecharts for test case generation
  - Statecharts and OCL constraints (pre-, postconditions) as test oracle

Bank Account

<table>
<thead>
<tr>
<th>Account Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>status: int</td>
</tr>
<tr>
<td>balance: int</td>
</tr>
<tr>
<td>isActive: boolean</td>
</tr>
<tr>
<td>isBlocked: boolean</td>
</tr>
<tr>
<td>isClosed: boolean</td>
</tr>
<tr>
<td>getBalance: int</td>
</tr>
<tr>
<td>activate</td>
</tr>
<tr>
<td>block</td>
</tr>
<tr>
<td>unblock</td>
</tr>
<tr>
<td>close</td>
</tr>
<tr>
<td>deposit(amount: int)</td>
</tr>
<tr>
<td>withdraw(amount: int)</td>
</tr>
</tbody>
</table>

Diagram of Bank Account:

- open
- blocked
- active
- deposit
- withdraw
- activate
- block
- unblock
- close
- closed
Bank Account

\[\text{context } \text{Account::withdraw(amount:int)}\]
\[\text{pre: } \text{true}\]
\[\text{post: } \text{self.status}_{\text{pre}} = \text{self.ACTIVE} \implies \text{self.balance} = \text{self.balance}_{\text{pre}} - \text{amount}\]

Test Case Generation

- Combining sequence diagrams and statecharts
- Main information from sequence diagrams
  - Inter object communication (collaborations)
  - Typical message sequences
  - UML 2.0: negative sequences
- Additional information from statecharts (protocol state machines)
  - Object life cycle
  - Initialise test cases
  - Test oracles
Test Case Generation

- Protocol state machine
- Life cycle of objects
- Call events
- No associated actions
- Implicit preconditions
- Observer methods

Positive Test Cases

:Bank
a:Account
b:Account

open
activate
block
unblock
close
closed

deposit
withdraw

deposit
withdraw

activate
block
unblock
close
closed

withdraw
deposit
Positive Test Cases

- open
- activate
- block
- deposit
- withdraw
- active
- unblock
- close
- closed
- blocked

Negative Test Cases

- open
- activate
- block
- deposit
- withdraw
- active
- unblock
- close
- closed
- blocked

- Bank
- Account
- a:Account
- b:Account
- transfer
- withdraw
- deposit
Negative Test Cases

Dehla Sokenou - TU Berlin
UML-Based Testing
Using Sequence Diagrams, Statecharts and OCL Constraints

Test Oracle

- Combining statecharts and OCL constraints
  - Implicit pre- and postconditions from statecharts
  - Explicit pre- and postconditions from OCL constraints
  - Linked by a logical AND (UML 2.0 semantics)
Test Oracle

- Statecharts pre- and post conditions
  
  **context** Account::withdraw(amount:int)
  
  **pre:** self.isActive
  
  state invariant of state active
  
  **post:**
  
  - self.balance@pre >= 0 implies self.balance >= 0 or
  - self.balance < 0 and
  - self.balance@pre < 0 implies self.balance < 0
- **OCL constraints**
  
  context Account::withdraw(amount:int)
  
  pre: true
  
  post: self.balance =
    
    self.balance@pre - amount

- **Resulting conditions**
  
  context Account::withdraw(amount:int)
  
  pre: self.isActive (and true)
  
  post: (self.balance =
    
    self.balance@pre - amount)
  
  OCL

  and
  
  (self.balance@pre >= 0
  
  implies self.balance >= 0 or
  
  self.balance < 0) and
  
  (self.balance@pre < 0
  
  implies self.balance < 0))

  statechart
Aspect-Oriented Programming

- Extension of object-oriented programming
  - Non-invasive code integration
  - Code weaving: before, after or replace method
  - Privileged access
  - Quantification and obliviousness
- Object Teams
  - Dynamic aspect weaving (load-time)
  - Dynamic aspect activation
  - Team structure
  - Aspects as roles

Aspects Used for Testing

- Object Teams for integrating test code
  - Statechart as role of object under test
  - Teams for each statechart level
  - Dynamic aspects for statechart implementation
  - More teams for OCL constraints and logging
- Calculation of expected results at runtime
Summary

- Combination of sequence diagrams, statecharts and OCL constraints
  - Information collected from different views
  - Positive and negative test cases
  - Independent test oracle
  - Benefits from aspect-oriented code integration

Outlook

- Future work
  - Integration of additional UML diagram types
    - Class diagram
    - Activity diagram
    - Additional OCL constraints
  - Test Data
  - Aspects and testing, or better: testing aspects (?)