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

Dehla Sokenou
TU Berlin
Softwaretechnik
Overview of the Test System

- **UML model**
  - abstract test cases
  - test oracle (UML)
  - test oracle preparation
  - test oracle integration
  - test data generation
  - test data
  - test execution
  - executable test oracle
  - test driver
  - test verdict

Dehla Sokenou - TU Berlin - Softwaretechnik
A UML-Based Testing Approach
Agenda

• motivation
• UML-based testing
  • test case generation
  • test oracle
• aspects used for testing
• summary and outlook
Motivation (1)

- open issue: testing object-oriented systems
  - problems: lack of specification, test code integration
- UML widely used for modeling and specifying object oriented systems
  - artifacts created in the analysis and design phases provide a good foundation for model-based testing
  - different views are modeled by using different diagram types

➤ our idea: combining several diagram types for testing
- test case selection based on UML diagrams
- main information from sequence diagrams
- additional information from state diagrams (UML statecharts) and OCL constraints
Motivation (2)

- test code integration often expensive
  - test code needs privileged access to the SUT
  - version control

- our idea: using dynamic aspects for testing
  - code is integrated in non-invasive manner
  - aspects have privileged access to the adapted system
Example: Bank Account

<table>
<thead>
<tr>
<th>Account</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>

protocol state machine

- open
- blocked
- active
- closed

- activate
- block
- unblock
- close
- deposit
- withdraw
Protocol State Machines

- life cycle of objects
- call events
- no associated actions
- implicit preconditions
- observer methods
Test Case Generation

⇒ based on sequence diagrams and UML statecharts
  • sequence diagrams
    • typical message sequences
    • communication between objects
  • statecharts (*protocol state machines*)
    • life cycle of objects
  • each sequence diagram = 1 test case
  • additional information from statecharts
    • initialization of test sequences
    • (test oracle)
Test Case Generation: Example

TF 1) x{new}; a{new}; a.activate; b{new}; b.activate; x.close
TF 2) x{new}; a{new}; a.activate; b{new}; b.activate; b.block; x.close
Test Oracle

- statecharts
  - implicit pre and post conditions
  - valid states and transitions
- OCL constraints
  - explicit pre and post conditions
- 2 variants of combination
  1. integration of OCL pre and post conditions into statecharts
  2. derivation of pre and post conditions from statecharts and combination with explicit OCL constraints
Test Oracle: Example (1)
Test Oracle: Example (2)

- protocol state machine

- OCL: pre and post condition

\[
\text{context} \\
\quad \text{Account::withdraw(amount:int)} \\
\quad \text{pre: } \text{true} \\
\quad \text{post: } \text{self.balance} = \text{self.balance@pre} - \text{amount}
\]
Test Oracle: Example (3)

- statechart: pre and post condition

```plaintext
context
Account::withdraw(amount:int)
pre:  self.isActive and (self.balance >= 0 or self.balance < 0)
post: ((self.balance@pre >= 0 implies self.balance >= 0 or self.balance < 0) and (self.balance@pre < 0 implies self.balance < 0)) and self.isActive
```

- OCL: pre and post condition

```plaintext
context
Account::withdraw(amount:int)
pre:  true
post: self.balance = self.balance@pre - amount
```
Test Oracle: Example (4)

- statechart: derivation of pre and post condition

**context** `Account::withdraw(amount:int)`

**pre**: `self.isActive and (self.balance >=0 or self.balance <0)`

**post**: `((self.balance@pre >= 0 implies self.balance >= 0 or self.balance < 0) and (self.balance@pre < 0 implies self.balance < 0)) and self.isActive`
Test Oracle: Example (5)

- resultant pre and post condition

**context** `Account::withdraw(amount:int)`

**pre:** `true and self.isActive` and ...

**post:** `((self.balance = 
  self.balance@pre - amount)
  and
  (self.balance@pre >= 0
  implies self.balance >= 0 or
  self.balance < 0)
  and
  (self.balance@pre < 0
  implies self.balance < 0)))
  and self.isActive`

```
resultant pre and post condition
```

```
context Account::withdraw(amount:int)

pre: true and self.isActive and ...

post: ((self.balance = 
  self.balance@pre - amount)
  and
  (self.balance@pre >= 0
  implies self.balance >= 0 or
  self.balance < 0)
  and
  (self.balance@pre < 0
  implies self.balance < 0)))
  and self.isActive
```

```
Test Oracle: Example (5)
```

```
resultant pre and post condition
```

```
context Account::withdraw(amount:int)

pre: true and self.isActive and ...

post: ((self.balance = 
  self.balance@pre - amount)
  and
  (self.balance@pre >= 0
  implies self.balance >= 0 or
  self.balance < 0)
  and
  (self.balance@pre < 0
  implies self.balance < 0)))
  and self.isActive
```
Test Code Integration (1)

- integration of test oracles into the SUT
  - aspect-oriented language: Object Teams
  - generation of executable statecharts
  - compilation of OCL constraints
- advantages
  - source and byte code of SUT not changed
  - aspects as roles with own state
  - tight coupling between aspect and role object
    - observer pattern already implemented (method calls are forwarded to aspect)
    - privileged access to the SUT
Test Code Integration (2)

- executable statechart with Object Teams
  - statechart as role of object under test
  - one team for each statechart level
  - dynamic aspect activation for statechart hierarchy implementation

- more teams for OCL constraints and logging
Test Code Integration: Example

team class Account_OCL {

class Account_Role playedBy Account {

    Account obj_AT_PRE;

    abstract boolean isActive(); isActive -> isActive; /* CallOutBinding */

    ... // also for clone and other query methods

    pre_withdraw <- before withdraw; /* CallInBinding */

    post_withdraw <- after withdraw;

    void pre_withdraw(int amount) { /* Implementation */

        obj_AT_PRE = clone();

        if (!pre) { // test failed }

    }

    void post_withdraw(int amount) { ... }

}
Summary

• combination of different diagram types
  • test case generation from sequences and statecharts
  • test oracle derivation from statecharts and OCL constraints
• information collected from different views
• independent test oracle
  • easy extension by using other diagram types
• aspect-oriented integration of test oracle
  • non-invasive integration
  • privileged access
Outlook

- integration of additional UML diagram types
  - class diagram
  - activity diagram
  - additional OCL constraints (beside pre, post conditions, invariants)
- derivation of test data from UML models
- use of efficient techniques
  - e.g. DresdenOCL
- industrial case study