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

UML-Based Testing

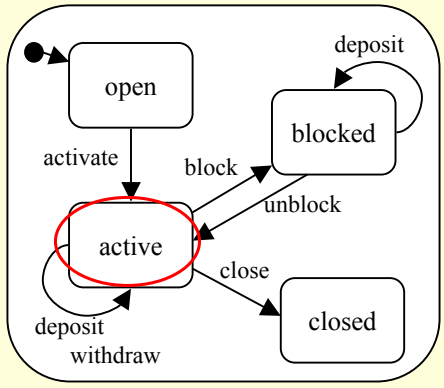
view	**diagram type**	**test use**
static	class diagram	preparation, configuration, oracle, test data
	object diagram	configuration
	management diagrams	
dynamic	interaction diagrams	test cases
	activity diagram	
	state diagram	oracle, test cases, test data
	use case diagram	coverage
static	OCL	oracle, test data

UML-Based Testing

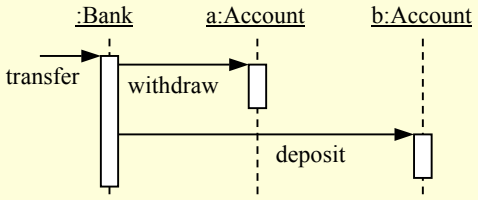
- Diagram types have different test relevance
  - Dynamic diagrams and OCL good foundation for testing
- Our approach combines
  - Interaction diagrams and statecharts for test case generation
  - Statecharts and OCL constraints (pre-, postconditions) as test oracle

Bank Account

Account
status: int balance: int
isActive: boolean isBlocked: boolean isClosed: boolean getBalance: int activate block unblock close deposit(amount: int) withdraw(amount: int)



Bank Account



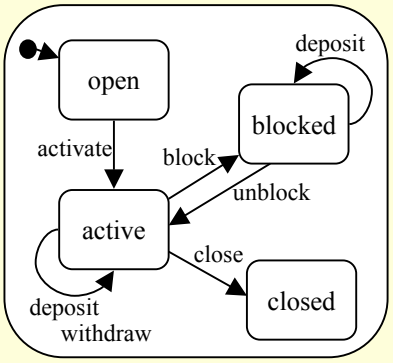
context Account::withdraw(amount:int)
pre: true
post: self.status@pre = self.ACTIVE implies
self.balance = self.balance@pre - amount

Test Case Generation

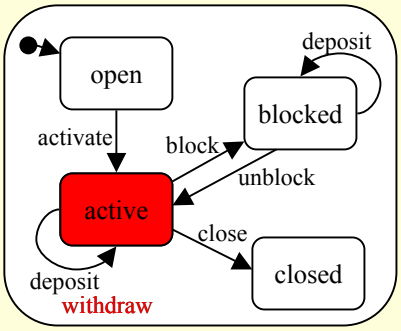
- Combining sequence diagrams and statecharts
  - Main information from sequence diagrams
    - Inter object communication (collaborations)
    - Typical message sequences
    - UML 2.0: negative sequences
  - Additional information from statecharts (protocol state machines)
    - Object life cycle
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

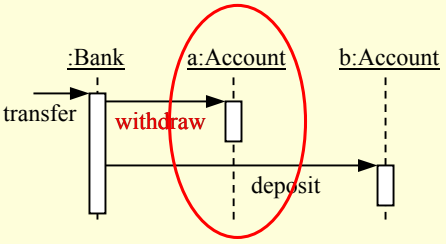
Test Case Generation

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

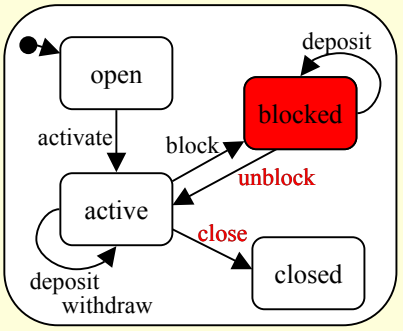


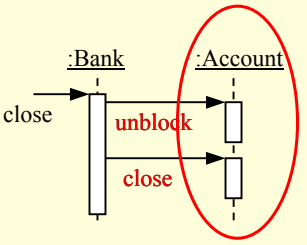
Positive Test Cases



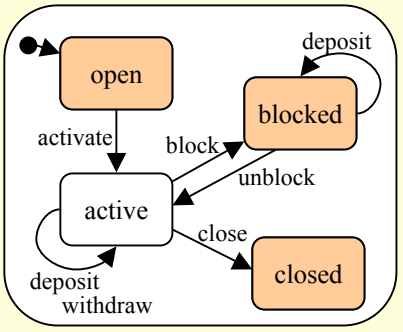


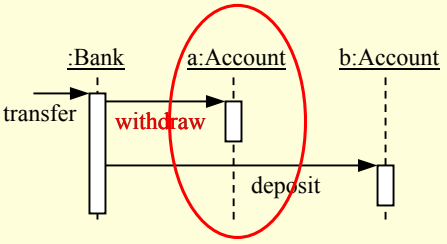
Positive Test Cases



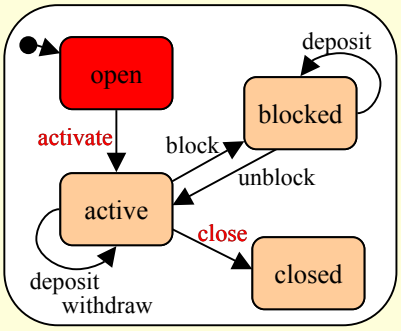


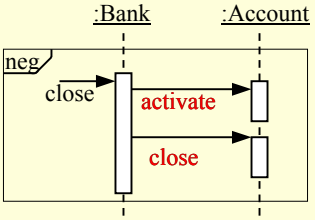
Negative Test Cases





Negative Test Cases

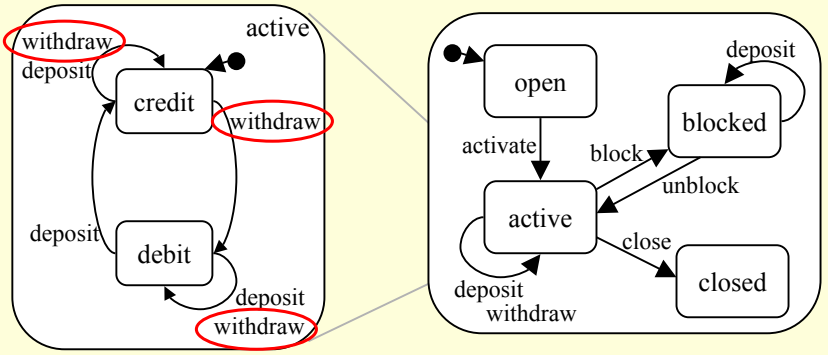




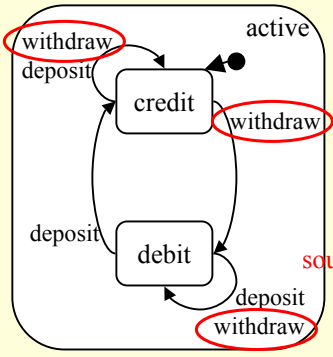
Test Oracle

- 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



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

source state *implies self.balance >= 0) or*

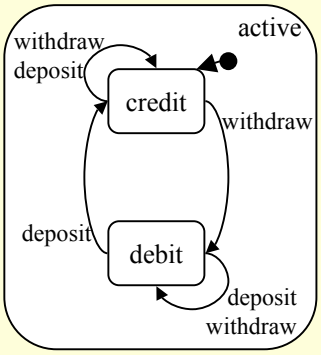
(self.balance < 0) and

target state

(self.balance@pre < 0

implies self.balance < 0)

Test Oracle



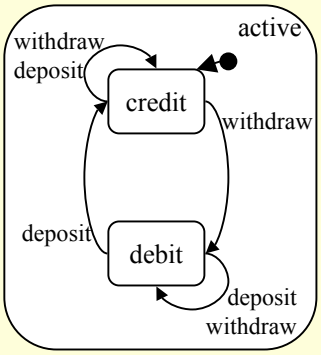
- OCL constraints

context Account::withdraw(amount:int)

pre: true

post: self.balance =
self.balance@pre - amount

Test Oracle



- Resulting conditions

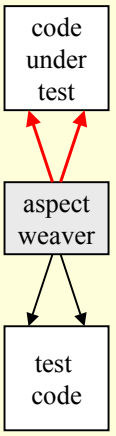
context Account::withdraw(amount:int)

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 } statechart
(self.balance@pre < 0
implies self.balance < 0))

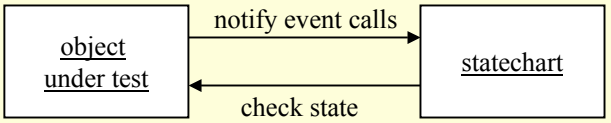
Aspect-Oriented Programming

- 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 (?)