Testing Functional Requirements in UML Activity Diagrams

www.modelexecution.org

Stefan Mijatov, Tanja Mayerhofer, Philip Langer, and Gerti Kappel

Business Informatics Group
Institute of Software Technology and Interactive Systems
Vienna University of Technology
Favoritenstraße 9-11/188-3, 1040 Vienna, Austria
phone: +43 (1) 58801 - 18804 (secretary), fax: +43 (1) 58801 - 18896
office@big.tuwien.ac.at, www.big.tuwien.ac.at
Motivation Model Driven Engineering (MDE)

Goals of MDE
1. Cope with **complexity** through abstraction
2. Increase **productivity** through automation
3. Increase **quality** through early **model analysis**

Increase quality through early model analysis

- Models are the central development artifacts
  - Models are the design, implementation, and documentation of software systems
- Quality of the software systems equates to the quality of the models
  - Any defect not detected at model level will be propagated to the code level
- Methods, techniques, and tools for developing high-quality models are crucial

→ Testing functional requirements in UML activity diagrams
Motivation

Unified Modeling Language (UML)
- Most widely adopted general-purpose modeling language in MDE\(^1\)
- Critique: No precise semantics

Foundational UML (fUML)
- Precise specification of behavioral semantics of foundational UML subset
- **Syntax**: Subset of UML defined with MOF-based metamodel
  - **Structural modeling**: Class diagrams (class, property, association, data type, etc.)
  - **Behavioral modeling**: Activity diagrams (activity, control nodes, actions, etc.)
- **Semantics**: Formal semantics and virtual machine
  - **Translational semantics** defined with first-order logic formalism Process Specification Language (PSL)
  - **Operational semantics** (virtual machine VM) defined with fUML itself
  - **UML is turning into a programming language**\(^2\)

---


Motivating Example

ATM

+ withdraw(Card, int, int) : boolean
+ deposit(Card, int, int) : boolean
+ startTransaction() : void
+ endTransaction() : void

completed Transactions: 0..*
+ current Transaction: 0..1

Transaction

- number : int

Card

- number : int
- pin : int
+ validatePin(int) : boolean

Account

- number : int
- balance : int
+ makeDeposit(int) : boolean
+ makeWithdrawal(int) : boolean

Record

- number : int
- amount : int
- timeStamp : TimeStamp

Deposit

Withdrawal

0..1

ATM::withdraw

card : Card

pin : Integer

amount : Integer

readAccount

Card::validatePin

pin

target

isPinValid

success

[true]

[false]

successFalse

result

target

mergeEnd

ATM::endTransaction

CallBehavior

splitCard

CallOperation

Account::makeWithdrawal

amount

success

readAccount

CallOperation

Card::validatePin

pin

target

isPinValid

success

[true]

[false]

successFalse

result

ATM::startTransaction

CallBehavior

ATM::withdraw

CallOperation

Card::validatePin

isPinValid

success

[true]

[false]

successFalse

result

mergeEnd

ATM::endTransaction

CallBehavior
Motivating Example

Functional Requirements (FR) for Successful Withdrawal

FR1  The pin has to be validated before the actual withdrawal is performed.

FR2  The account's balance has to be reduced by the provided amount of money.

FR3  The activity should return true indicating a successful withdrawal.

FR4  When the withdrawal is started, a new transaction should be created; once it is completed, the transaction should be ended and recorded.

FR5  After the completion of the withdrawal, the balance of the account should be equal to the difference between the sum of all recorded deposits and the sum of all recorded withdrawals.
Testing Framework

Test Input Data
- Input values for input parameters
- Initial system state
- Example: Amount of money to be withdrawn from account with balance € 100

Execution Order Validation
- Validation of chronological order in which activity nodes are executed
- Example: FR1 The pin has to be validated before the actual withdrawal is performed.

Input / Output Validation
- Validation of expected output for given input
- Example: FR3 The activity should return true indicating a successful withdrawal.

State Validation
- Validation of the runtime state during execution of activity under test
- Example: FR4 When the withdrawal is started, a new transaction should be created; once it is completed, the transaction should be ended and recorded.

Test Specification Language & Test Interpreter
Testing Framework

Overview

fUML Model
- Classes
- Activities

Test Suite
- Test Cases
  - test t1 activity a1 {
    assertState before
    x = 300
  }
- Test Scenario

Test Specification Language

Model Execution → Execution Trace → Test Evaluation

Test Interpreter

Test Verdict
- t1
  - assertState
  - failure

fUML VM RI$^1$ + Trace Extension$^2$

---


Testing Framework

Test Input Data: Test Scenarios

Syntax

TestScenario := \texttt{scenario} name \{ (TestObject | TestLink)* \}

TestObject := \texttt{object} name : UML::Class \{
(UML::Property = UML::ValueSpecification)* \}

TestLink := \texttt{link} UML::Association \{ UML::Property = TestObject
UML::Property = TestObject\}

Example

\texttt{scenario} atmTestData \{
\texttt{object} atmTD : ATM \{
\texttt{object} cardTD : Card \{ pin = 1985; \}
\texttt{object} accountTD : Account \{ balance = 100; \}
\texttt{object} depositTD : Deposit \{ amount = 100; \}
\texttt{link} card_account \{ account = accountTD; card = cardTD; \}
\texttt{link} account_record \{ account = accountTD; records = depositTD; \}
\}

## Testing Framework

### Test: Test Case

#### Syntax

Test := `test` name `activity` UML::Activity

(UML::ActivityParameterNode = (TestObject | UML::ValueSpecification))*

(on TestObject)? { ... }

#### Example

```plaintext
`test` atmTestSuccessfulWithdrawal `activity` ATM.withdraw (card=cardTD, pin=1985, amount=100) on atmTD {
...
}
```

![Diagram of ATM withdrawal test case]
Testing Framework Execution Order Validation: Order Assertions

Syntax
OrderAssertion := assertOrder ( * | _ | UML::ActivityNode ) *

Example
FR1 The pin has to be validated before the actual withdrawal is performed.

```plaintext
test atmTestSuccessfulWithdrawal activity ATM.withdraw (card=cardTD, pin=1985, amount=100) on atmTD {
   assertOrder *, validatePin , *, makeWithdrawal, *
}
```
Test Evaluation

- **Challenge**
  - fUML VM delivers execution trace from one activity execution only
  - Considering execution order captured in single trace is insufficient in case of concurrency (false positives)

- **Solution**
  - Calculate concurrent branches based on input / output dependencies of executed activity nodes (control flow and data flow)
  - Capture dependencies in adjacency matrix
  - Evaluate order assertion based on adjacency matrix
FR1 The pin has to be validated before the actual withdrawal is performed.

```plaintext
assertOrder *, validatePin, *, makeWithdrawal, *
```
FR1 The pin has to be validated before the actual withdrawal is performed.

(assertOrder *, validatePin, *, makeWithdrawal);
Challenges

- Precise selection of relevant runtime states
- Evaluation of complex conditions on selected runtime states

Solution

- Introduction of temporal operators and quantifiers for precise selection of runtime states (temporal expressions)
- Integration of OCL for specifying and evaluating complex conditions on runtime states (state expressions)

Syntax

StateAssertion := assertState TemporalExpression { StateExpression* }

Selection of runtime states  Validation of runtime states
Syntax: Temporal Expressions

TemporalExpression := (always | sometimes | immediately | eventually) 
   (after | until) (action UML::Action | constraint OCL::OclExpression)

Temporal Operators

after  ... selects states after the execution of action / fulfillment of condition
until  ... selects states before the execution of action / fulfillment of condition

Quantifiers

always  ... property has to hold for all selected states
sometimes ... property has to hold in some of the selected states
immediately ... property has to hold in the first / last selected state
eventually ... property has to hold from some of the selected states on

Syntax: State Expressions

StateExpression := check OCL::OclExpression {on UML::ObjectNode}?
Example
FR4 When the withdrawal is started, a new transaction should be created; once it is completed, the transaction should be ended and recorded.

assertState eventually after constraint TransactionCreated {
    check TransactionEnded, TransactionRecorded;
}

countext ATM
exp TransactionCreated : currentTransaction <> null
exp TransactionEnded : currentTransaction = null
exp TransactionRecorded : completedTransactions -> size() = 1

assertState eventually after constraint atm.currentTransaction <> null {
    check atm.currentTransaction = null, completedTransactions -> size() = 1;
}
Testing Framework

State Validation: State Assertions

assertState eventually after constraint atm.currentTransaction <> null {
    check atm.currentTransaction = null, completedTransactions -> size() = 1;
}
Evaluation

User Study

Criteria

1. **Ease of use**: How easy is it to use the testing framework for testing UML activity diagrams?

2. **Usefulness**: Are test results useful for detecting and correcting defects in UML activity diagrams?

Setup

1. **Introduction** to fUML and testing framework

2. **Self-assessment** of experience with UML, OCL, and unit testing

3. **Testing tasks**
   - a) **Ease of use**: Write test cases for validating predefined functional requirements in given and correct UML activity diagrams
   - b) **Usefulness**: Resolve defects in UML activity diagrams based on given test cases and test results

4. **Questionnaire** on experienced ease of use and usefulness
**Testing Task 1: Writing Test Cases (Ease of Use)**

- Minor issues with some concepts of test specification language
  - E.g., purpose of test scenarios, jokers in order assertions
- Most issues could be resolved by trial and error, and consultation of introductory material

→ Gentle learning curve of test specification language
→ Improvements of concrete syntax and editor support (e.g., validation)

**Testing Task 2: Resolving Defects Based on Test Results (Usefulness)**

- Understanding test cases: Participants were able to identify tested functional requirements
- Understanding test results: 59% of defects were resolved on average

→ Indication that test results are useful for locating and resolving defects
→ Improvements of visualization of test results (supplementary debugging support is important!)
Questionnaire on experienced ease of use and usefulness

- Write test cases
- Read test cases
- Read test results
- Correct activity diagrams

0 1 2 3 4 5 6 7 8

very easy easy medium hard very hard

Evaluation Results
Summary

- **Validation and verification** of models are essential in MDE
- **Testing** framework for validating the fulfillment of **functional requirements** in **UML activity diagrams**
  - Test scenarios for defining test input data
  - Order assertions for validating execution order of actions
  - State assertions for validating runtime state of system during execution, and output of activities and actions

Future Work

- Extensions of testing framework with **additional testing capabilities** (e.g., conditions on differences between distinct runtime states)
- Improvement of **feedback on test results** (e.g., visualization of runtime states, failure cause analysis)
- Support for **further executable sublanguages of UML**
Thank you!

Model Execution Based on fUML

www.modelexecution.org