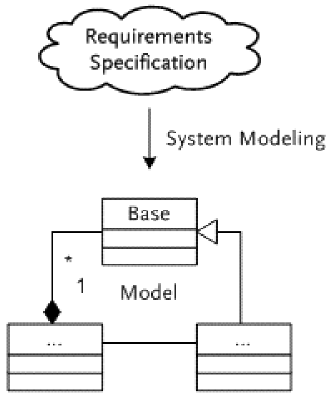
Cause-Effect Graphs for Test Models Based on UML and OCL

Stephan Weißleder, Dehla Sokenou

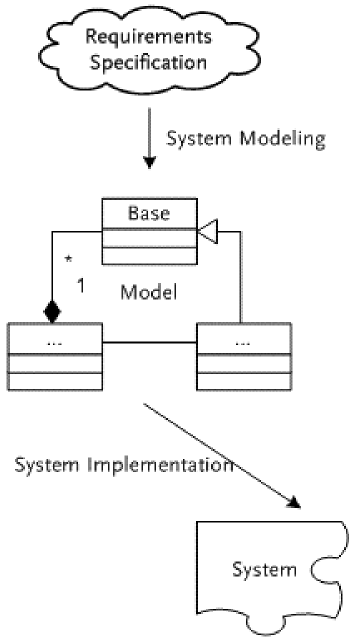
Table of Content

Model-Based Testing

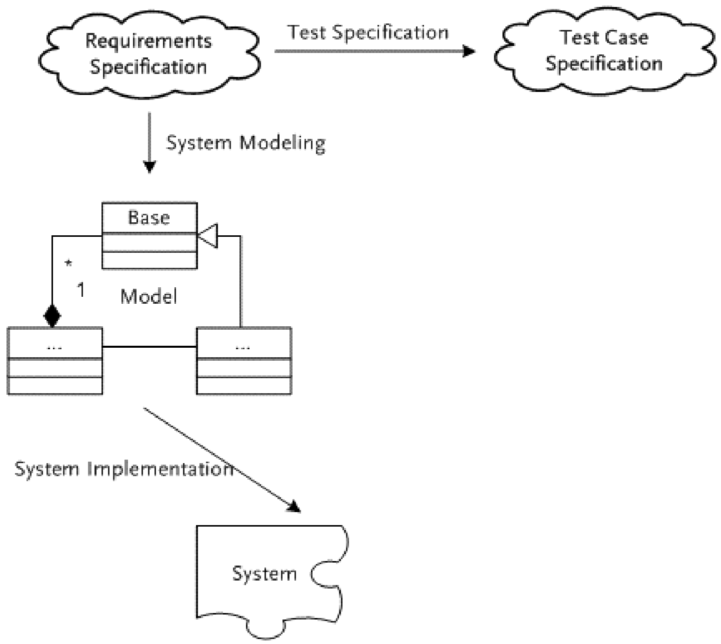
Model-Based Testing



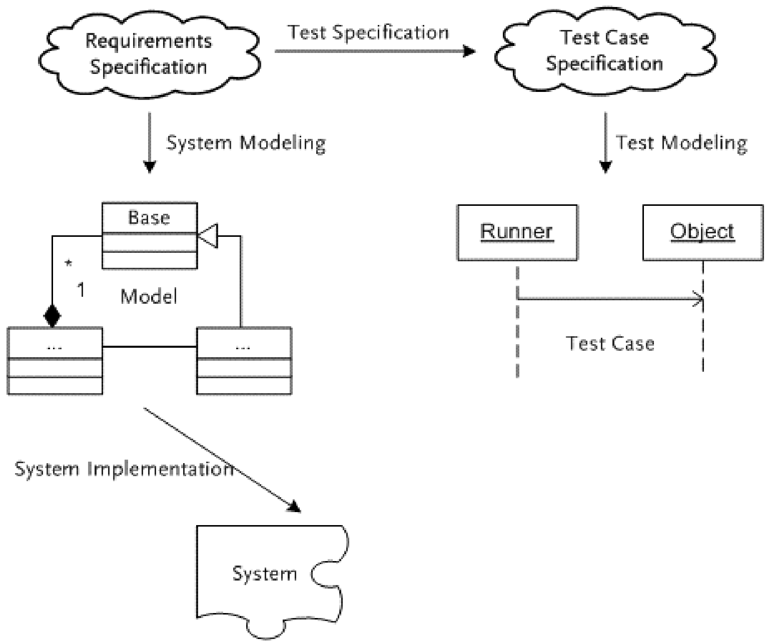
Model-Based Testing



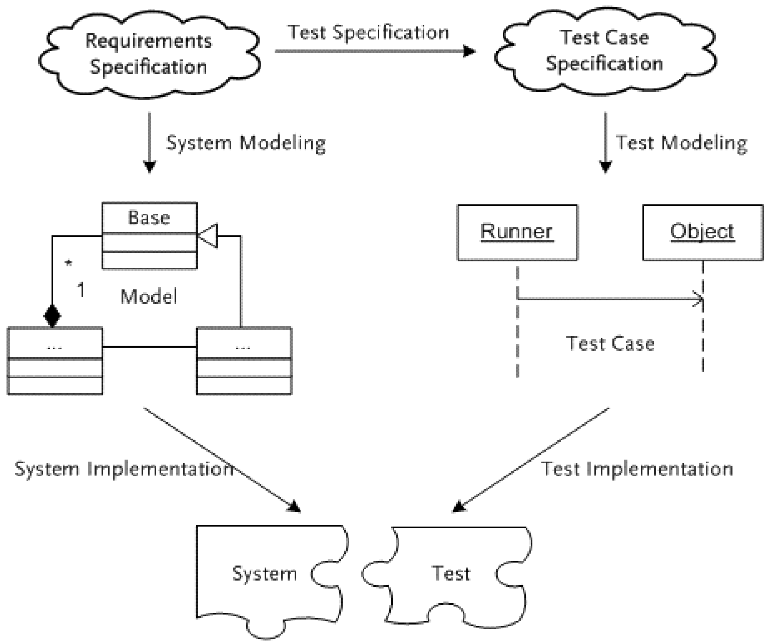
Model-Based Testing



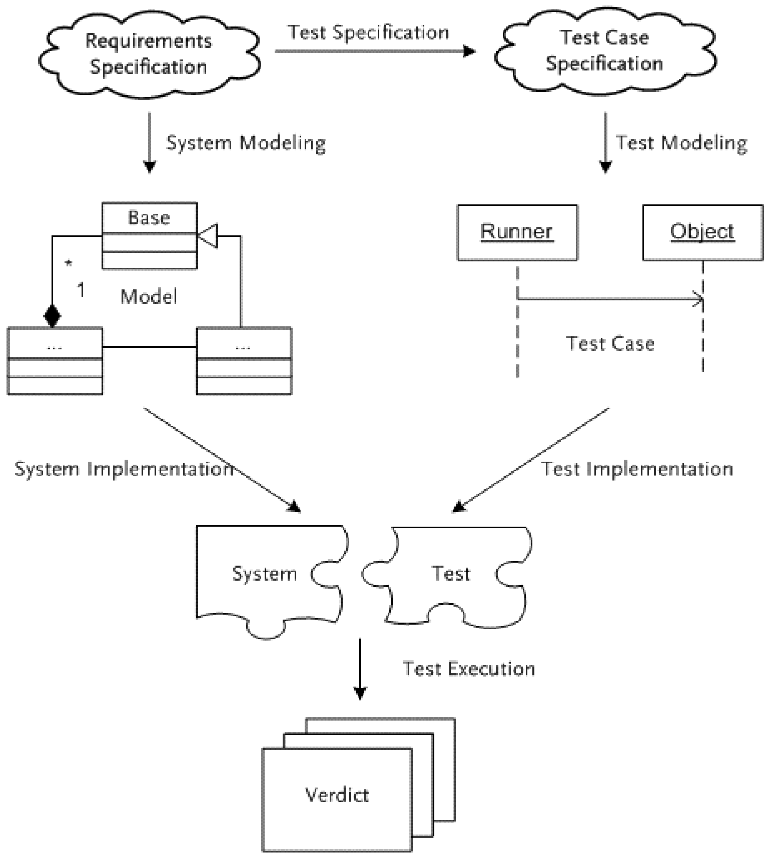
Model-Based Testing



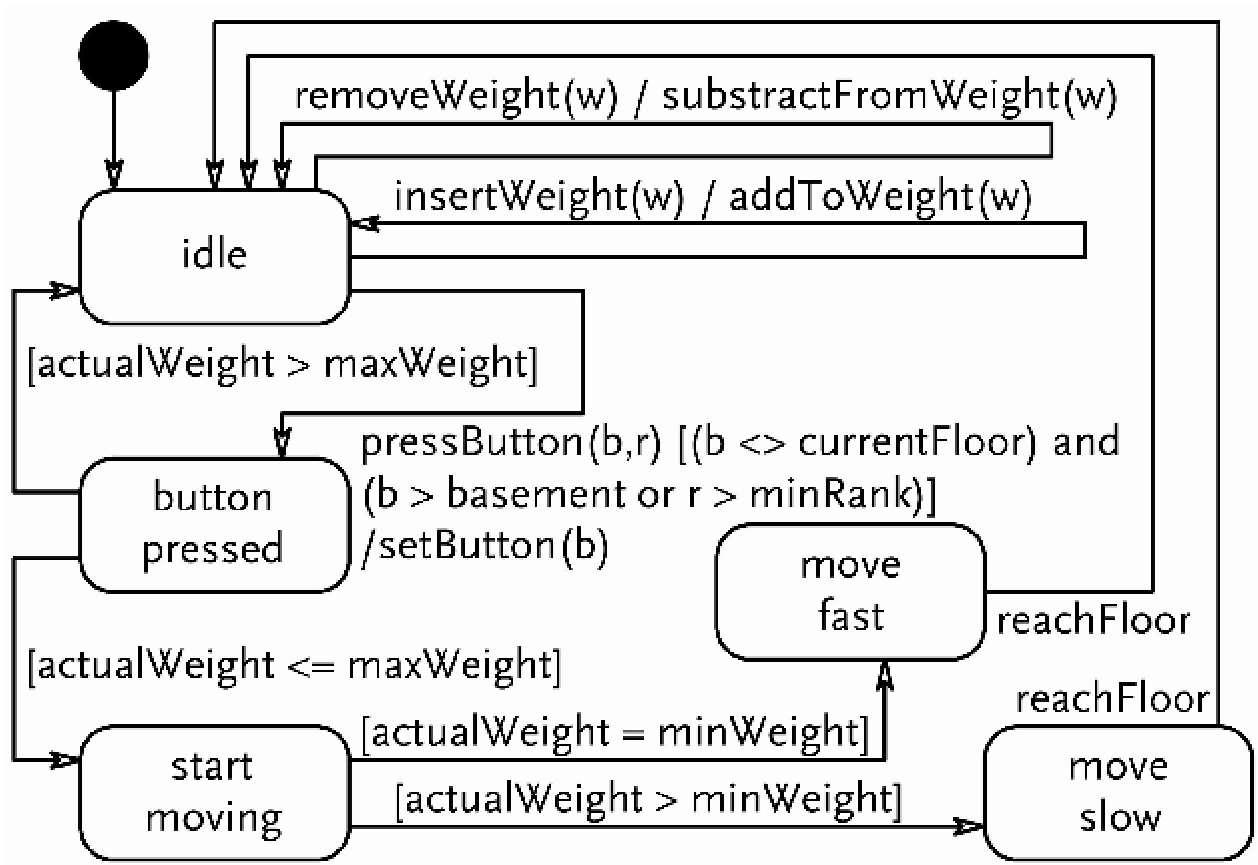
Model-Based Testing



Model-Based Testing

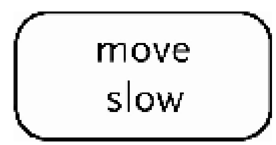


Example of a Freight Elevator



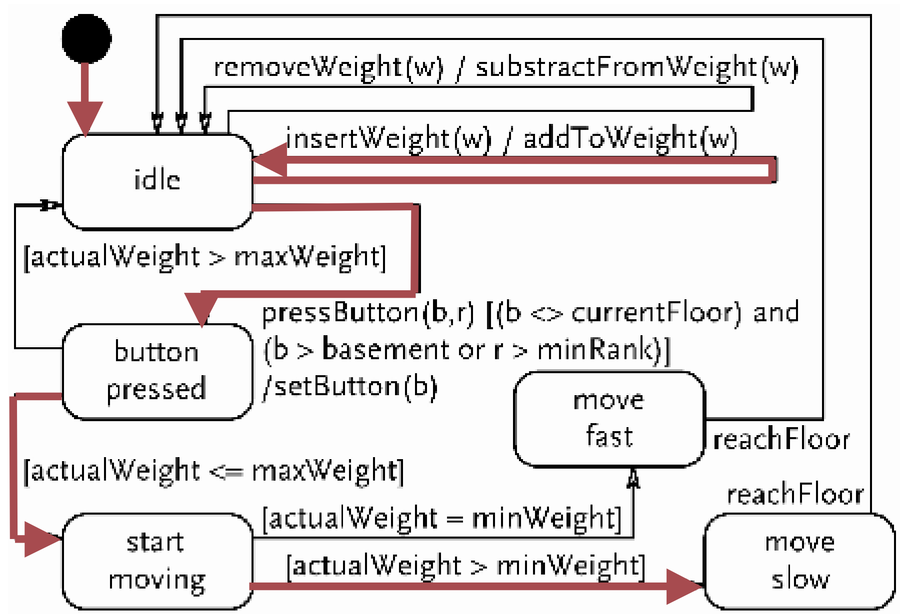
Automatic Test Case Generation from UML and OCL

- Model-based test generation from behaviour specifications:
  - Search forward from initial node
  - Search backward from test goals
- Test goal:
  - Coverage criterion applied to a concrete model
  - Example: one state for *All-States*
- Generate abstract test case
  - Find a path
- Generate concrete test case
  - Find concrete input values



Automatic Test Case Generation from UML and OCL

- Generation of test cases:
  - Path from initial node to test goal contains conditions (e.g. OCL)
  - Due to conditions not each found path is feasible
  - Consequence: include conditions into search algorithm
  - Deal with the relations between OCL conditions along the path



Automatic Test Case Generation from UML and OCL

- Generation of Test Cases:
  - Classify all variables used in the OCL expressions
    - Which variables can change?
  - Algorithm - for each guard:
    - try to find postconditions that influence the result of the guard
    - Combine guard and postcondition to a new condition
    - If there are changeable variables in the condition: continue search
  - Basic Idea:
    - Transform conditions on system attributes into conditions on input parameters
    - Use them as input partitions

Quality of Test Suites

- Satisfying coverage criteria:
  - Algorithm generates tests that satisfy all guard conditions
  - Several coverage criteria also need negative cases:
    - Condition Coverage
    - Decision Coverage
    - MC/DC
  - Consequence:
    - Add test goals for transformed guard conditions
    - Find test cases that satisfy them

[actualWeight > minWeight] → [actualWeight > minWeight]

[actualWeight > minWeight] → [actualWeight <= minWeight]

Classification of OCL Elements

- Why classify OCL elements at all?

Postcondition:

X = Y@pre

X := Y
Y := X

X := Z
Y := Z

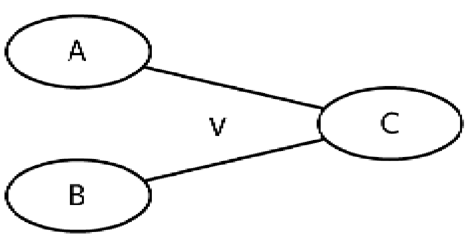
Classification of OCL Elements

- Examples for Condition Transformations
  - Guard: [actualWeight > minWeight]
    (passive) (passive)
  - Post: actualWeight = actualWeight@pre + w
    (active) (passive) (passive)
  - New: actualWeight@pre + w > minWeight
  - Satisfaction of guard condition is now influenced by input parameters
  - Is this relationship easily visible?
  - Supporting readability by using cause-effect-graphs?

Introduction to Cause-Effect-Graphs

- Graphical representation of
  - Logical expressions
  - Logical dependencies
- Easier to read than textual representation

- Example: C = A v B

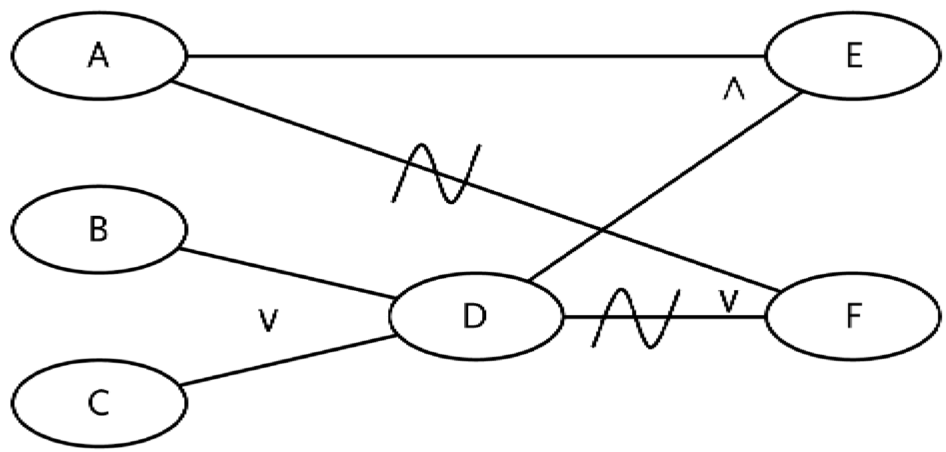


Introduction to Cause-Effect-Graphs

- (b <> currentFloor) and (b > basement or r > minRank)

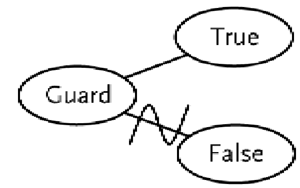
  A B C

- E = true, F = false

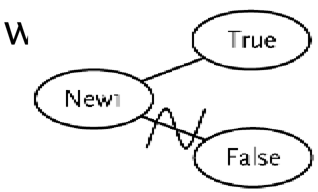


Using Cause-Effect-Graphs to clarify OCL Relationships

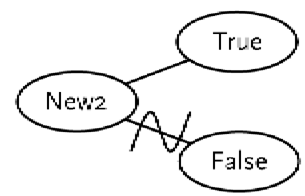
- Expressing relationships between OCL expressions
- Guard: [actualWeight > minWeight]



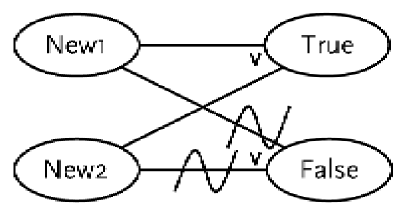
- Relation 1
  - Post of addToWeight: actualWeight = actualWeight@pre + w
  - New1: actualWeight@pre + w > minWeight



- Relation 2
  - Post of substractFromWeight: actualWeight = actualWeight@pre – w
  - New2: actualWeight@pre – w > minWeight



- Combination



Discussion

- Improved readability by using cause-effect-graphs?
- Case studies needed
- Cause-Effect-Graphs are related to the test case
  - The order of input stimuli has effects on the cause-effect-graph
  - Examples:
    - Execute addToWeight AND removeFromWeight
    - Execute addToWeight two times
  - Result:
    - Adapt causes depending on the structure of the cause-effect-graph
- Include information about input events in cause-effect-graph?
- Same principle used in prototype ParTeG (http://parteg.sf.net)