Interpack 2011

Model – Based Testing of Embedded Control Systems

Prof. Dr. Holger Schlingloff
Fraunhofer FIRST
Humboldt University, Berlin
Holger Schlingloff

- Research Director Software Quality in the Fraunhofer FIRST, Embedded Systems Department
- Professor of Specification, Verification and Testing Theory at Humboldt University, Berlin
Structure of this Talk

- Model-based Testing
- Deriving Models from Requirements
- Generating Tests from Models
- Improvement of Test Suites
Model-based Testing

Def. (IEC 61508): dynamic Testing = deliberate und systematic execution of software and/or hardware to show the presence of required and absence of unwanted behavior defined with respect to requirements systematic, goal orientation different ways to proceed
code-based testing: systematic and goals given by the code
model-based testing: systematic and goals given by the specification, i.e. the formalized requirements
Formalization of Requirements

There are different levels of abstraction and formalization:

- user / system requirement specifications
- use case descriptions
- algebraic or logical formula
- class diagrams with pre / postconditions
- UML state diagrams
- finite automata
- Matlab/Simulink MFiles
- code / pseudo-code, ...

- generation, execution and evaluation of tests with respect to different kinds of models
Structure of this Talk

• Model-based Testing
• Deriving Models from Requirements
• Generating Tests from Models
• Improvement of Test Suites
Use Case Descriptions

- often formulated in natural language (e.g. in DOORS-data base)
- natural language is informal, imprecise, and ambiguous. It cannot be executed by a machine.

⇒ methodology for the transformation of textual use case descriptions into formal models
Use Case Validator
Use Case Validator
Structure of this Talk

- Model-based Testing
- Deriving Models from Requirements
- Generating Tests from Models
- Improvement of Test Suites
Example Test Generation

- testing goal:
  - state "move slow"
- start in this state, backward search
Example Test Generation

- backward step
- transition-condition:
  - (actualWeight > minWeight)
- list of conditions:
  - (actualWeight > minWeight)
Example Test Generation

• backward step
• transition-condition:
  - (actualWeight <= maxWeight)
• list of conditions:
  - (actualWeight > minWeight)
  - (actualWeight <= maxWeight)
Example Test Generation

- backward step
- transition-condition:
  - \((b <> \text{currentFloor} \text{ and } (b > \text{basement} \text{ or } r > \text{minRank}))\)
- can be realized via pressButton\(b, r\)
- list of conditions:
  - \((\text{actualWeight} > \text{minWeight})\)
  - \((\text{actualWeight} \leq \text{maxWeight})\)
Example Test Generation

- backtracking
- additional transition insertWeight(w)
- conditions can be satisfied by an appropriate choice of w
Realisierung

- ParTeG: „Partition Test Generator“
  - tool prototype
  - generates JUnit test cases from XML models
  - several coverage criteria can be selected
  - integrates with Eclipse environment
  - SourceForge

- used in various industrial and academic projects
Structure of this Talk

- Model-based Testing
- Deriving Models from Requirements
- Generating Tests from Models
- Improvement of Test Suites
A Safety protocol for Industrial Automatization

- **PROFIsafe**
  - international field bus safety standard
  - safe transmission layer above the transport layer
    - host device
    - guarantees one channel safe transmission via unreliable medium
    - sequence numbering, watchdog, timeouts, CRC-Checks
  - reference testing suite needed
PROFIsafe Slave State Machine
Coverage Improvement by Evolutionary Search

- Enhancement of existing test suite via genetic algorithm
Summary

Model-based Testing
Deriving Models from Requirements
Generating Tests from Models
Improvement of Test Suites

Thank you for your attention!