Approaches to Improve System Dependability – From Formal Verification to Model-Based Testing

Andreas Ulrich, Peter Amthor, Marlon Vieira
Siemens AG, Corporate Technology, CT SE/SCR
andreas.ulrich@siemens.com

Siemens Corporate Technology
Present in all leading markets and technology hot spots
Focus of Work in Dependability Engineering

- **Attributes**
  - Availability
  - Reliability
  - Safety
  - Confidentiality
  - Integrity
  - Maintainability

- **Means**
  - Fault prevention
  - Fault tolerance
  - Fault removal
  - Fault forecasting

- **Threats**
  - Faults
  - Errors
  - Failures

**Our focus**
- Dependability Competence Team at Siemens CT SE

[source: J.-C. Laprie et al., 2000]
Fault Prevention by Model Driven Design and Formal Verification

Application Domain

Requirements → Informal Design Model → Formal Model (refined) → Formal Verification → Results

- Properties to be checked
- Results

- Correct Model

Formal Verification

- Formal model
  - Typically extracted manually from an informal model and requirements
  - But possible reuse of models from model-driven design
    - Matlab/Simulink
    - Lustre, Esterel (Scade)
    - Statecharts, UML models (Rhapsody etc.)
  - Requires transformation to the input language of a model-checker

- Set of properties
  - What properties to be checked?
    - Structural properties, reachability
    - Derived from requirements
  - Requires in-depth system knowledge and knowledge in formal languages (e.g. LTL) → hard!
Example: Verification of the PROFIsafe Protocol

PROFIsafe Protocol Architecture

- **Host Application Process**
  - F-Host

- **Slave Application Process**
  - F-Input Slave

**Failsafe Communication**

Grey Channel

- Peers F-Field Device
- F-Gateway
- Master-Slave Assignment
- Other Safety Bus

**Coexistence of standard and failsafe communication**

- PG/ES with secure access e.g. Firewall
- TCP/IP

**PG/ES with secure access e.g. Firewall**

**Engineering Tool**

- Profibus DP
- Standard-Host/PLC
- Standard-I/O
- Repeater

**Emergency push buttons**

**F-Host/FPLC**

**F-Input Slave**

**F-Field Device**

**F-Actuator**

**F-Sensor**

**F-Actuator**

**F-Host/FPLC**

**F-Input Slave**

**F-Field Device**

**F-Actuator**

**F-Sensor**

**F-Actuator**

**F-Host/FPLC**

**F-Input Slave**

**F-Field Device**

**F-Actuator**

**F-Sensor**

**F-Actuator**

**F-Host/FPLC**

**F-Input Slave**

**F-Field Device**

**F-Actuator**

**F-Sensor**

**F-Actuator**

**F-Host/FPLC**

**F-Input Slave**

**F-Field Device**

**F-Actuator**

**F-Sensor**

**F-Actuator**

**F-Host/FPLC**

**F-Input Slave**

**F-Field Device**

**F-Actuator**

**F-Sensor**

**F-Actuator**
PROFIsafe Modeling Approach

1. PROFIsafe Statechart

2. Refined Statechart

3. MC Input Model (in-house MC used)

PROFIsafe Properties

- **Structural properties**
  - Absence of livelocks and deadlocks
  - External inputs can be handled in all states
  - Deterministic behavior

- **Specific properties**
  - F-host activates FV after Timeout
  - F-host activates FV after CRC Fault
  - F-slave activates FV after Timeout
  - F-slave activates FV after CRC Fault

- **Possible improvements**
  - Check CRC in initial operation of F-host
  - Clarify how F-slave should handle status bits 2 and 3

Fault identified
Correct Model – What Next?

- Generation of production code from verified model
- How feasible is the approach?
  - Automatic code generation is still a challenge, e.g. in the embedded domain on special hardware
  - Verified model mostly domain-specific, i.e. no general-purpose code generator
  - Efficiency of auto-generated code?
  - Generated code must run in an unknown, i.e. not verified, environment
    - If environment is unknown, how sure can one be that the verification results are preserved?
  - Product certification does not like auto-code generation

- Is there an alternative to production code generation?
  - Yes, generation of test code, model-based testing!

Proving vs. Testing

Beware of bugs in the above code;
I have only proved it correct, not tried it.

[Donald E. Knuth, 1977]
PROFIsafe Test Case Generation (Fault Removal)

Test Sequence #1
1. rcv[pv][0][0][nok]
2. resume
3. send[1][0][1][0]
4. rcv[pv][0][1][ok]
5. resume
6. send[1][0][1][1]
7. rcv[pv][0][2][ok]
8. resume
9. send[1][0][0][2]
10. rcv[pv][0][3][ok]
11. resume
12. send[1][0][0][3]
13. rcv[pv][0]
14. resume
15. send[0][0][0][4]
16. ...

PROFIsafe Project Results

- **Test cases**
  - Serve as compliance checks for PROFIsafe product suppliers
  - Identified errors in PROFIsafe reference implementation
  - Accepted by TÜV Süd Deutschland for certification

- **Formal verification**
  - Helps identify ambiguities in system requirements and the design model
  - Supports a clean documentation of the design
  - Requires close communication link between domain experts and verification experts
  - Still an expensive approach
Model-Based Testing – Evolution

- Design and test process
  - Model-driven development vs.
  - Test-driven development
- Ingredients of MBT
  - Formal model, e.g. UML + semantics
  - Test generation algorithm
  - Coverage criterion
- Our focus
  - MBT approaches based on test models mainly
  - UML based techniques, U2TP

Test Generation: The TDE/UML Tool – Workflow

1. System Specification
   - Use Case Specification
   - Flow of Events
   - Business Rules
2. Annotations
3. UML Models
4. Test Management
5. Application SUT
6. Executing/Verifying Test Scripts
   - Feedback
   - Successful Validation
   - Developers
   - Test Executor
   - Capturing Snippets
   - Test Management
   - UML Models
Modeling in TDE/UML – The MS Word Example

Modeling of a GUI pop-up window...

... in a UML activity diagram + annotations

TDE/UML – The NYC Subway Project

- Project at Siemens Transportation (TS)
  - Deployment of a MBT approach in system testing of the NYC Subway Railcom project

- Contribution
  - Modeling of about 300 system requirements in UML in Rational Rose
  - Generate system tests in IEEE 829 format
  - Creating and running executable test scripts in Rational XDE Tester

- Benefits to the customer
  - Currently about 130 test cases generated (about 200 expected)
  - Modeling helps uncover incomplete and/or inconsistent specifications
  - Cost of maintenance is reduced due to a systematic and repeatable test approach
The NYC Railcom Project – TDE/UML Approach

**Coverage**: round-trip criteria

**Refinement**: all refinements

**Data variations**: all choices

---

### TDE/UML Summary

- Supports a model-based approach for testing
- Generation of test cases for manual and/or automatic execution
- Current projects demonstrate the usefulness of the approach
  - Decreased effort for test maintenance
  - Increased notion of requirement coverage during test creation/execution
  - Decreased overall time for test creation
- Model-based testing must be introduced as a service
  - Models are domain dependent
  - Transformation from informal requirements to formal models requires experts
  - Lower entry level required compared to formal verification
How to Improve the Penetration of Formal Methods?

→ “Invisible” formal methods may offer high rewards at low/moderate efforts

Invisible Formal Methods in Practice – Further Work

- Some new commercial tools deploy this principle
  - Support of extended source code analysis
    - Polyspace
    - Intel Thread Checker
  - Support of test case generation
    - Reactis (for Simulink Stateflow)

- Other (similar) projects in this context at Siemens CT SE
  - Fault diagnosis from communication traces of a distributed system using formal verification (SPIN)
    - Model reconstruction from traces
    - Library of predefined properties (application dependent, e.g. UTRAN)
  - Validation of (manually derived) test suites, e.g. quality, coverage