Need an integrated solution to address both challenges of modeling and validation?

Free morning session

« A unique tool for modeling & test generation guided by properties »

Thursday, June 12 | 09:00-12:00
@Nano-INNOV [Palaiseau]

PragmaDev and the CEA LIST present the first version of their integrated tool for **modeling and optimized test generation** driven by **coverage and properties**

**Preliminary agenda**

- PragmaDev modeling technology
- CEA LIST validation/verification technology
- Use case feedbacks
- Demonstration of tool integration

**Free registration**

**Access map**

To unsubscribe, send an email
PragmaDev - Formal modelling technology

PragmaDev

• French SME,
• Created in 2001 by 2 two experts in modelling tools and languages
• Since creation dedicated to the development of a modelling tool for the development of Event driven software.
• Since 2013 focused on the delivery of a full Model Based testing solution
References

- Universities: ENST, Polytechnica Bucarest, Telecom Beijing...
- Distributors: United States, Asia, Europe...

Several Projects with big accounts

- Alcatel-Lucent → Exoticus
  Started in 2005, finished in 2009
- THALES →
  Started in 2012, finished in 2014
- PragmaList →
  Started in 2013
Issues

“The software content is doubling about every two years. The sheer volume is making it increasingly difficult for QA and test teams to keep up with traditional tools and processes.”


Never forget the 1:10:100 rule

Source: James Martin study

“What software development costs only comprise a portion of the total cost of software ownership. However, the development process itself has a significant impact on total cost of ownership.”

Total Cost of Ownership: Development is (only) Job One by Daniel D. Galorath - June 2008.

What modelling can solve

- Focus first on the What (Req) instead of focusing on the How (code),
- Modelling is about Communication and Documentation,
- Legibility of large systems gets critical. Would you build a house without drawing detailed plans?
- Get control over productivity,
- Improve quality.
Requirements for a good modelling language

- The abstract model must be platform independent, as its name states.
- The abstract model must be translatable to an implementation platform.
- For that purpose, the abstract model is based on a virtual machine offering:
  - Some **basic services**.
  - A strong enough **semantic**.

Modelling levels

- Event driven technology
- Three levels of modelling in the tool:
  - Informal,
  - Semi-formal,
  - Formal.
- Based on international standards.
- Integrated in one framework.
Model simulator

A graphical debugger for fully formal models (and TTCN-3 test cases).

• Set breakpoints and step in the model,
• Dynamic traces.

Model coverage

• Graphical model coverage analysis
• Merge feature
Prototyping interface

- Connects automatically to the simulator or the debugger.
- Knows about the model inputs and outputs.

Standard testing language

- Relies on basic services
  - Messages
  - Procedures
  - Timers
  - Parallel execution
- Based on TTCN-3 international standard:
  - Data types definitions or ASN.1,
  - Templates definitions,
  - Test cases,
  - Verdict,
  - Execution control.
TTCN-3 support

- Textual language
- Simulator with Test manager
- C++ code generator
- TTCN-3 to MSC generation
- MSC to TTCN-3 generation
- TTCN-3 generation from a property on the model (Verimag)
- TTCN-3 generation based on model coverage (to come)

Continuous integration

 specification

 implementation

 validation testing

 integration testing

 unit testing

 simulation

 execution
Model checking

- Partnership with specialized labs:
  - Exhaustive simulation,
  - Symbolic resolution.
- Properties:
  - Model coverage,
  - Static or dynamic property:
    - Property verification,
    - Test objectives.

Reference testing

Diagram:
- Model
  - Simulation Execution
  - Traces
  - Test objectives
  - Coverage
- Requirements
- Tests
- Result of PragmaList
Property Sequence Chart

- PRESTO European project:
  - Functional property verification.
  - Non functional property verification.
  - Free tool: PragmaDev Tracer

Conclusion

- Three levels of modelling:
  - Informal,
  - Semi-formal,
  - Formal.
- Tools to:
  - Document,
  - Simulate,
  - Validate,
  - Test.
- Based on international standards.
- Integrated in ESA Taste framework.
Roadmap

- **Short term:**
  - PragmaList use case validation.
  - MacOS X version.

- **Mid term:**
  - PragmaList: new features added after first use cases
  - New third party tool support (debuger, RTOS).
  - Market Specialization (Railway, M2M, IoT).

- **Long term:**
  - Z.104 support.
  - New ergonomics.
CS RTDS use case:
Managing complexity in Airbus Air Traffic Control

Julien Honoré
ATC project leader at CS

AGENDA

CS
  History
  ATC SDL models and workshop
  Complexity
  Example
  Coverage need
  Conclusion
CS Group: Overview

- CS, designer, integrator & operator of mission critical systems
  - Prime contractor for turnkey systems, featuring innovation and performance
  - Active across the entire value chain: Consulting, Design, Build, Run
  - Culture of expertise & innovation

CS, Avionics department informations

- Activities scope
  - System engineering activities (70%)
  - Formal specification in embedded SW (30%)
- Multi customers: AIRBUS, ATR, Eurocopter, Honeywell, Rockwell Collins, SCLE, ...
- Resources: 65 engineers: average of 5 years experience in system engineering
- Working experience
  - Supporting major customers in the definition and integration of complex systems
    - System & I/F definition, High-level requirement, A/C installation requirement, change management
    - Formal specification in SDL and Verification & Validation activities
    - Supplier follow up (activities monitoring, follow up, ...)
    - Avionics maintainability studies
    - V&V activities (definition, coordination, test support, test result analysis, ...)
    - Contribution to the Aircraft certification
    - ....
**History**

- Design and verification of SDL models for more than 10 years

- Realization of Air Traffic Control (ATC) onboard applications for all Airbus aircrafts
  - Communication between aircraft and ground stations
  - Communication with other onboard equipments
  - Interaction with captain / first officer through dedicated HMIs

- Need to get SDL specifications coverage to answer to DO178C objectives

**ATC SDL Models and workshop**

- Design and verification of SDL Z.100 models
  - Architecture
  - Behaviour (SDL state machines)
  - Use of qualified automatic code generator to produce C code

- Specification of operators and frontiers
  - Operators hand-coded in C and called by SDL

- Verification of ATC application
  - SDL simulator coupled with CS workshop (partially automated tests and verifications)
  - On host verification of system specifications
ATC SDL Models and workshop

Set of tools developed
- To simulate environment (Cockpit HMIs, Ground stations, onboard equipments)
- To check some rules on models
- To automate verification

Use of operators
- To realize complex operations (when SDL is not suitable or for efficiency);
- To acquire data from other onboard systems:
  - Altitude, Flight number, etc.
  - For simulation purpose, use of simulated operators where those data are managed in text files

Coverage information retrieved in verification stages and merged together

Complexity

Use of complex models
- ~60 blocks, ~100 processes, ~2 000 procedures, ~2 000 input symbols, ~4 000 output symbols, ~25 000 tasks, ~320 000 C loc

Data types
- Use of complex imbricated structured types
- Use of charstring type (up to 4 000 characters)
- No use of reals
- Use of timers and « duration » data type

Important number of contexts/scenarii
- Many interfaces with environment
  - Through SDL signals at frontier
  - Through C operators
- Many decisions imbrications
1. Ground center sends encoded connexion request message sig1(..., charstring msg, ...)
2. Process « p1 » decodes the message
3. « p1 » transmits the message to « p2 » (if no decoding error) sig2(..., struct msg, ...)
4. « p2 » finds valid parameters and creates an instance of « p3 »
5. « p2 » transmits the message to « p3 » sig3(..., struct msg, ...)
6. « p3 » retrieves the current time and generates text to display through two dedicated operators
7. « p3 » sends those information to « p4 » sig4(..., charstring text2display, ...)
8. « p4 » generates the frame to be displayed to pilot using operators for charstring functions
9. « p4 » sends the frame to cockpit display sig4(..., charstring frame, ...)

**Context/scenario:**
- Ground station sends a well-encoded message with valid parameters
- There is no time acquisition problem
- ...

**Coverage need**

- Need to identify missing contexts and scenarios to reach 100% coverage
  - It is quite complex to identify scenarios that will completely cover the entire system
  - The use of manually hand-coded C operators increases this complexity
- Need to have a tool to help in this activity
  - Tool shall be able to process the complexity of our models
    - Size
    - Explosion of number of contexts
  - Scenarios/contexts shall be expressed in order to be understandable for SDL designers
    - To ease creation of functional verification scenarios
  - Scenarios/contexts shall consider
    - the entire system as a black box
    - operators that modify SDL process variables
Conclusion

→ Recently adopted DO-178C/ED-12C considers model based approach
  › Reaching 100% test coverage can reduce certification activities (reviews of model...)
  › 100% model coverage will be more and more expected by certification authorities

→ Coverage criteria
  › All decisions branches of all ATC SDL model processes

Thank you!

→ Questions?
Automatic Test Generation: the DIVERSITY Approach

Alain Faivre
Arnault Lapitre
Our Approach (1/3)
Scenario Generation for Model Validation

Our Approach (2/3)
Test Generation for Black-Box Testing
Our Approach (3/3)

Model-Based Testing

Modeling

Requirements

Real system / Test Bench

Implementation / Compiling

DIVERSITY - xLIA

Model:
- concurrent / communicating automata,
- dataflow language

Coverage criteria:
- states / transitions
- MC/DC

Structural constraints:
- nb of tests,
- size of a test

Coverage rate

Test suite
Symbolic simulation of the model:

- Defines **symbolic behaviours**, i.e. **equivalence classes** of numerical behaviours of the system.

- Represented as a tree.
Symbolic simulation of the model:

- Defines **symbolic behaviours**, i.e. **equivalence classes** of numerical behaviours of the system.
- Represented as a tree.
- Each path = a distinct symbolic behaviour.

Example of symbolic behaviour for a lift: the cabin moves upwards.

Example of symbolic behaviour for a lift: the cabin goes from 1st to 6th floor.
Moreover, the tool enables to:

- Detect inconsistency among data types,
- Detect dead locks,
- Detect dead parts of the model,
- Check syntactic properties of the model (to be defined with the user),
- ...

DIVERSITY – Coverage Criteria

- Structural stop criteria
- Transition / State coverage for Stateflows,
- DC, MC/DC,
- Inclusion criterion
- Logical formula coverage,
- ...

All previous coverage criteria may be used to generate test cases in a particular context defined by properties expressed with the help of the model variables.
MC/DC Criterion
For each logical subsystem or function, and each input $e_i$, generates test cases showing that $e_i$ independently impacts the output.
**Logical Formula Coverage**
Generates a test suite verifying a given set of properties.

E.g.
- mode = nominal (or mode = degraded)
- 10 ≤ speed ≤ 100
- nb_of_passengers > 4
- ...

**Hit or Jump** strategy: to compute one or more symbolic behaviours of the system corresponding to the test purposes.

Objective: to cover the path $t_1.t_2...t_n$ of a function $F_i$

Calculus Strategy: Hit or Jump

HIT: execution of the transition $t_1$
Objective: to cover the path $t_1, t_2, \ldots, t_n$ of a function $F_i$
Objective: to cover the path $t_1, t_2, \ldots, t_n$ of a function $F_i$. JUMP: random selection of a leaf node.
Objective: to cover the path $t_1, t_2, \ldots, t_n$ of a function $F_i$

Calculus Strategy: Hit or Jump

HIT: execution of the transition $t_2$
Objective: to cover the path $t_1, t_2, \ldots, t_n$ of a function $F_i$

Stop: when we find a path which ends with the execution of $t_n$
Objective: offer coverage criteria complying with industrial standards:
• DO-178B/C
• ISO 26262
• EN 50128
• Common criteria
• …

For example – ISO 26262
Table 14 — Structural coverage metrics at the software unit level

<table>
<thead>
<tr>
<th>METHODS</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a Statement coverage</td>
<td>A</td>
</tr>
<tr>
<td>1b Branch coverage</td>
<td>B</td>
</tr>
<tr>
<td>1c MC/DC (Modified Condition/Decision Coverage)</td>
<td>C</td>
</tr>
</tbody>
</table>

Covered by DIVERSITY

Conclusion

DIVERSITY
- Test generation from models
- Deterministic strategy => more selective than random strategies
- Modular tool architecture => easy to add new coverage criteria

Collaboration with industrial partners:
- To assist them in building models for automation
- To add new criteria adapted to industrial needs and standards
New ad hoc criteria for:
  • Specific features of modeling languages
    (stateflow, dataflow, execution and communication semantics, …)
  • Needs and constraints of industrials, standards requirements

• Test of non deterministic systems
• Test of component oriented systems
• Model refinement and test case generation
• Test of distributed architecture
• …
The project in four steps.

- **Step 1 : SDL to xLIA traduction rules** :
  - Write the traduction rules to convert SDL to xLIA.

- **Step 2 : SDL to xLIA translator** :
  - Write the xLIA generator from an SDL model.

- **Step 3 : Diversity adaptation to support SDL semantic** :
  - Work on SDL communication semantic,
  - Work on SDL timer semantic.

- **Step 4 : TTCN-3 formats output generation** :
  - TTCN-3 test cases formatting to be supported by RTDS.
Four types of verification

- **Code coverage**: To generate the minimum number of test cases that cover all transitions.
- **Transition**: To generate a test case that covers a specific transition in the SDL model.
- **Property**: To generate the test cases verifying a static property (process state, variable value, …).
- **Observer**: To generate the test cases verifying a dynamic property (succession of action or temporal rules). A dynamic property is defined as a state machine called observer.
**Demonstration**

- Exploration time is always the same (10 seconds) whatever are the message parameter ranges.

**Verimag IF toolbox**

- Exhaustive exploration
- Exploration time depends on message parameter range.

<table>
<thead>
<tr>
<th>Digit range</th>
<th>0..1</th>
<th>0..2</th>
<th>0..3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Card range</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0..1</td>
<td>13</td>
<td>126</td>
<td>721</td>
</tr>
<tr>
<td>0..2</td>
<td>38</td>
<td>316</td>
<td>2169</td>
</tr>
<tr>
<td>0..3</td>
<td>64</td>
<td>650</td>
<td>28234</td>
</tr>
</tbody>
</table>

*Time to explore the model in seconds*
• Integrated tool chain

• Non dedicated model

• Efficient symbolic kernel
  ➢ Test automation
  ➢ Reduce the number of test cases
  ➢ Early in the development process