

Verification of Model-Based Software – Challenges from a Certification Perspective

Devesh Bhatt

Honeywell International Inc.
Aerospace Advanced Technology Labs
Golden Valley, Minneapolis, MN

Caltech Workshop on Verification and Validation
Pasadena, California
September 23-24, 2009

Verification of Model-Based Software – Challenges from a Certification Perspective

Current practice within Honeywell in Certifying Avionics Systems

High-Level Requirements → MATLAB Model (Design) → Source Code → Executable Object Code
(up to 20x-50x cost and time savings in commercial avionics programs such as flight controls)

- RTCA DO-178B Objectives:
 1. Verify that design meets high-level requirements (functional) and derived safety requirements
 2. Verify that object code is robust and meets all elements of the design (current cost driver)
- Home grown model static analysis and test generation tools (Objective 2)
 - applied on thousands of real MATLAB Simulink/Stateflow models (tools are qualified for use in DO-178B processes) – tool maturity at production level
- Getting 95-100% automation in commercial avionics - flight controls, engine controls, perimeter controls
 - Model → DO-178B certification artifacts (analysis, tests, results)

Verification of Model-Based Software – Challenges from a Certification Perspective

- Models are getting bigger, more complex
 - New functionality- adaptive controls, more interaction with other controllers, “efficiency”
 - Additionally, use of low-level modeling constructs (in practice) causes loss of “design intent”
 - Too much intermingling of continuous and discrete, direct patching up, tuning of the models
 - Need proper abstractions for top-down system design

Two factors exacerbate this

1. Verification tools automate analysis/checking at this low level, so no incentive for modelers to improve the design factoring/abstractions
2. Whatever we learned as “good design practice” and “software engineering” is not applicable when modeling tools such as MATLAB don’t readily enable this
 - Should disable the “Simulate” button until after a design review
 - Static Analysis++ (range, temporal, other semantics) should be used at early design stages

Verification of Model-Based Software – Challenges from a Certification Perspective

- Need to use higher-level semantics in design and verification
 - E.g., In Simulink – use of higher-level blocks and patterns, driven by control semantics
- Partial automation of verification does not help the cost/benefit argument
 - Use combination of static analysis, model checking, test generation
- Can design be factored (“aspects?”) such that the critical parts can be verified to a greater degree than non-critical?
 - Need formal, automated argument that the non-critical aspects will not have undesired impact on the behavior
 - Can we treat software “bugs” same as hardware component failures that get handled in some way in the computation of total reliability
- High-Level requirements – no good methods to specify them for automation of downstream verification that the design model meets requirements
 - Many application sub-domains (e.g., maintenance, built-in tests, I/O processing) are not “control laws”
 - DO-178C (SC205) recommends a “formal” method be avoided for specifying high-level requirements