Verification and Validation of Flight Critical Systems (VVFCS)

Area 2 (SSAT 4.1.3) – Integrated Distributed Systems
Area 4 (SSAT 4.1.4) – Software Intensive Systems

Kevin.Driscoll@Honeywell.com
Brendan.Hall@Honeywell.com
Kevin.Schweiker@Honeywell.com
Devesh.Bhatt@honeywell.com
• NASA is sponsoring a multiple-area multi-year program for verification and validation of flight critical systems.

• Objective
  - Provide advanced analytical, architectural, and testing capabilities to enable sound assurance of safety-critical properties

"Validated, proactive solutions for ensuring safety in flight and operations"
• Program to mature processes and tools, by creating:
  - Advanced analytical, architectural, and testing capabilities
  - Comprehensive collection of re-usable models
  - Approaches enabling objective engineering trade-offs to resolve debates about “best” approach

• Motivation
  - Integrated systems are becoming more complex
  - Next Gen systems will be even harder
  - Often a gap between formal theory and real-world systems
    - E.g., Byzantine fault tolerance is often over-looked
    - On other hand, systems designed for worst-case theoretical modes of failure can be overly brittle
    - Need better modeling technology to focus attention on what really matters

• Team
  - Prime: Honeywell
  - Subs: SRI and WW Technology Group

• Phases
  - Phase I: One Year — September 2010 to September 2011
  - Phase II: Two Years — September 2011 to September 2013
Main focus on communication networks
- Why are data networks so important?
  - Network(s) form the backbone of a system
  - The design of the network(s) becomes an approximation for the system architecture
  - In the absence of system architects, the network designers become the architects
- As “glue” for a fault tolerant system, the network must be more dependable than any component

Some Key Results
- Maturation of Architecture Analysis and Design Language (AADL)
- Discovered an edge-case in TTEthernet
  - Able to fix the SAE 6802 standard before its ratification
  - Able to fix NASA Orion CEV ASIC design before “cast in silicon”
- System-level test generation for distributed architecture
  - Full MC/DC protocol coverage
• Initial AADL modeling of diverse set of networks
  - SAFEbus (a backplane bus based on self-checking pairs)
  - TTP/C (a time-triggered bus protocol using simplex nodes)
  - SPIDER (a voter-based Byzantine-tolerance broadcast network)
  - BRAIN (a braided ring using high integrity message forwarding)

• PRISM probabilistic modeling of the SPIDER broadcast protocol

• Model-driven distributed test generation

• EDICT modeling derived from some of the AADL models and explored "out-of-band" error propagation

• A framework for relating properties in architectural models to control software models to support an end-to-end assurance case
• **Year 2 – Extending models to applications**
  - Triplex high-integrity control case-studies
    - Asynchronous / Time Triggered Architectures
    - Homogeneous / Heterogeneous Networks
    - Voted / Masked fault tolerance strategies
  - Technologies
    - Formal analysis of architecture behavior and key safety properties
    - Integrated formal analysis of continuous and discrete systems
    - Integration of AADL error and behavioral modeling

• **Year 3 – Extending models for system-of-systems**
  - Looking for some good system-of-systems examples
Idealized architecture validation automated architectural derived test generation

Formal Architecture Modeling and Analysis Technology

Analytical Studies

Industrial Practice (Architecture validation)

Case Studies

Mine data and experience from real qualification tests of real architecture

Lesson’s Learned

SAE Working Groups

ARP4754A, ARP4761, AIR6110, AS5506A

ARP4754A: Guidelines for Development of Civil Aircraft and Systems
ARP4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment
AIR6110: Contiguous Aircraft/System Development Process Example
AS5506A: Architecture Analysis & Design Language (AADL)
Async Case Study
- Generic Ethernet / AFDX

Plant Model
Simulink

Application (Control Law)
Simulink

Redundancy Management (3x)
Asymetry Management
- Abstractions / Behavioral Models

Protocol (Encrypted Wrap)
- Abstractions / behavioral

Integrated Architectural Model

EDICT

Formal Models
(PVS, SAL, Hybrid SAL)

DO-160
Environmental Conditions and Test Procedures for Airborne Equipment

EDICT

 application package

Test Generation
HybridSAL-ATG

Performance Models

Component Failure Models

Test Generation

Evidence Integration
EDICT, ETB

System & Safety Properties
(e.g. Force Fight, HW3)

5506A, 5506/1, 5506/2, 5506/3

AADL Working Groups

ARP 4754, ARP 4761, ARP 5107, AIR 6110

VVFCS
Area 2 – Phase II Async Case Study Influence Flow

Lessons Learned
An Evidential Tool Bus for Flight-Critical Software Systems

• We are developing a semantic framework for the end-to-end assurance of flight-critical software, specifically
  - Model-based design methodologies
  - Analysis capabilities based on powerful deductive tools
  - Formalized mathematical libraries for engineering complex systems
  - Compositional analysis of software-intensive systems

• The Evidential Tool Bus (ETB) is a platform for integrating multiple analysis capabilities into a unified assurance case

• Team
  - Prime Contractor SRI International: Shankar Natarajan and Sam Owre
    • shankar@csl.sri.com
  - Sub Contractor Honeywell: Devesh Bhatt, David Oglesby, Gabor Madl
    • devesh.bhatt@honeywell.com

Note: The remainder of this presentation contains Honeywell’s examples for model-based analysis tools interacting over ETB.
Each tool registers a set of *claims rules* with ETB that the tool can invoke to satisfy the claim: e.g., Yices_satisfiable, HiLiTE_range_bounds_check.

- Data inputs and outputs associated with a claim: values, files (hash), JSON objects (e.g., range bounds).
- A tool can make a *query* (proof obligation) that can be satisfied by another.
- The assurance case gets dynamically constructed: ETB invokes tools to satisfy claims – and new claim queries that get generated. (*Claims Table*)
**Benefit:** Reduced certification cost & cycle time; detect design problems early
Claim Examples:
“hilite_ModelDefects(modelname, signal_dict, all_defects, [ ])”
“hilite_CheckRangeBound(modelname, {signal_name, bound}, signal_dict, [ ])”
• Claims are uniquely identified with the specific version of files (hash) and value arguments
  - Development artifacts: requirements, models, source code, object code
  - Verification artifacts: verification properties, range bounds, model defects, tests, test results

• Versions of files are identified with hash and maintained in distributed repositories (e.g., git) at multiple ETB nodes.
  - E.g., presence of a new object code file can trigger a chain of claims and tool invocations to create tests from the corresponding version of model, static analysis, checking of model, etc.
    - Claims related to previous versions of these artifacts are irrelevant
Keeping track of changes, dependencies, and incremental verification

- Many artifacts go into a verification task (activity)
  - Input artifacts to the activity must have claims associated with specific versions of those artifacts
  - Info in files’ headers needs to be matched/verified to claim an artifact’s veracity
- **Current process**: detailed manual work instructions that trace artifacts and changes
- **ETB process**: claims and goals automatically generated and chained – change impact analysis can be automated by reverse chaining
  - E.g., presence of new object code can set off chain of claims related to source code, model, compiler

Composing a DO-178B/C verification objective from multiple claims

- A typical DO-178B/C verification objective has many sub parts which require different types of analyses and verification methods
- **Current process**: each objective (including all sub parts) is assigned to a specific team or tool; sometimes requires manual work/interaction to complete sub parts
- **ETB process**: goals for each sub part of the objective are automatically generated and can be assigned to different teams/tools that can produce claims for those goals.
• Dispositions of problems/observations found as a result of a verification activity
  - E.g., in model analysis and test generation, sometimes design defects and test coverage holes are observed
    - Each of these observations needs to be “disposed” in a combination of several ways: analysis of existing verification artifacts, supplemental analysis, design change.
    - Often, “dispositions” require manual analysis and generation of related artifacts
  - **Current process:** issue/problems reports exchanged among multiple design/verification teams, manual analysis work – delays and informal/undocumented assumptions
  - **ETB process:** queries can be automatically generated from “dispositions” returned by a tool’s analysis, analyses can generate claims to satisfy the queries, assumptions are formally documented and justified and can be extracted automatically by wrappers from artifacts
    - E.g., if a variable is not explicitly initialized in the model/code then a model analyzer tool generates a query that can be satisfied by a claim (made from HW model) that all variables are implicitly initialized to 0 in flash memory.