National Aeronautics and Space Administration

SSAT-3
Validation and Verification of Safety-Critical Integrated Distributed Systems

Kevin Driscoll
Fellow

2011 Annual Technical Meeting
May 10–12, 2011
St. Louis, MO

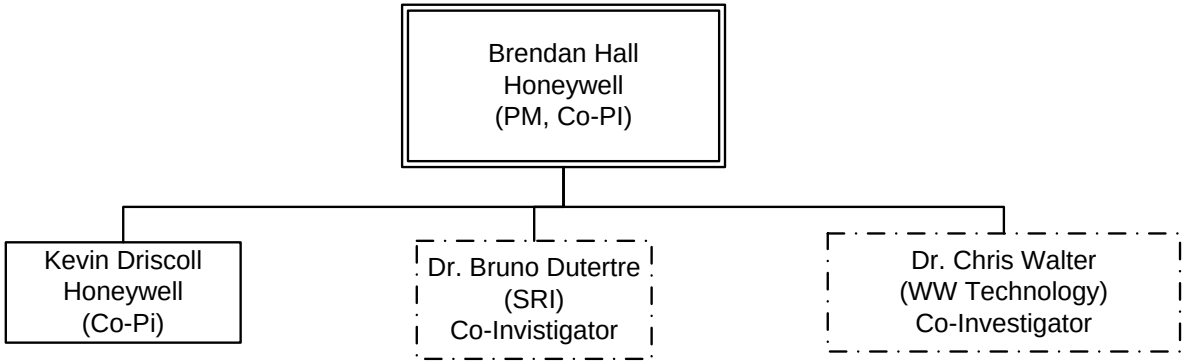
Background

- Motivation
  - Integrated Systems are becoming more complex
    - Hybrid Network Architectures
    - Mixed mode time-triggered/asynchronous systems
    - Interaction of plant and control
  - Next Gen Systems will be even harder
  - Currently, often a gap between formal theory and real-world systems
    - Byzantine fault tolerance often over-looked in real world systems
    - Systems designed for theoretical worse case failure modes can be overly brittle
  - We need better modeling technology to help focus attention on what really matters
- Goal
  - Provide advanced analytical, architectural, and testing capabilities to enable sound assurance of safety-critical properties for distributed systems of systems
  - Establish a comprehensive collection of re-usable models supporting the V&V of a broad array of distributed systems; enable objective engineering trade-offs to resolve debates about “best” approach
- Open Innovation
  - Evolve open standards such as Architecture Analysis Design Language (AADL)
  - Evolve “state-of-the-art” formal analysis and modeling tool chains

Program Objectives

- Develop analysis tools and techniques
  - For diverse types of distributed systems
    - Focus on the key element of distribution: The Data Network
  - Applicable to a range fault-tolerance frameworks
  - Techniques and models openly available and reusable
  - Publically available artifacts
- Investigate techniques modeling faults and error propagations
  - AADL and EDICT
  - Application to real-world IMA candidate architectures
- Investigate feasibility of integrating formal analysis and test generation
- Develop advanced modeling and analysis capabilities to address emerging trends in integrated distributed systems architectures
- Develop models of faults and fault propagation for (NextGen-related) systems-of-systems

The Team and Responsibilities



- Honeywell International Inc.
  - Fielded and research fault-tolerant architectures
  - Knowledge of actual failure modes and propagation mechanisms
  - AADL
- SRI International*
  - Verification and validation tools
    - Prototype Verification System (PVS), Symbolic Analysis Laboratory (SAL)
  - Evidential Tool Bench (tool integration framework)
- WW Technology Group*
  - Modeling and analysis with AADL
  - Error Detection Isolation Containment Types (EDICT) tool

* have demonstrations this evening

Tasks: Outline

- Models of Failures, Disturbances and Degradation
  - AADL and EDICT models
  - Approach for analyzing probabilistic models
- Communication and Distribution
  - Modeling Real-World Systems
  - Integrating Formal Analysis and Arguments
  - Investigating Test Generation from Formal Models
- New Systems Decompositions and Functional Integrators
  - Mixed synchronous/asynchronous systems
  - Fault propagation in dynamic topologies

Models of Failures, Disturbances and Degradation

- Taxonomy of failure modes and discovery process framework
  - Develop TRIZ style examination framework to elicit system failure potential
- Improve system and fault propagation modeling
  - Enable modeling and capture of environmental and common mode influences
- Develop probabilistic extensions to SAL model checking and analysis tool-kits
  - Allow probabilistic examination of formal models
    - What is the probability of my system failing in this mode?

Value of Formal Method Tools

In addition to the general definition of a tool's value being a labor saving device, what should a tool do?

- If it only regurgitates what you know, it's a waste. (A tool that requires such complete and detailed input that it is just the output in different form, is not an FM tool, it is a pretty printer.)
- If it confirms what you know by ensuring nothing has been overlooked, it has some value.
- The best value comes from a tool that tells you something you don't know (a counter example).
- There is negative value in a tool the re-enforces incorrect or (unknown to user) incomplete thinking.
- There is even greater negative value in a tool that instills incorrect or (unknown to user) incomplete thinking in the naïve user.

Bad tools can be dangerous.

The Byzantine Generals Problem

- A type of failure –
  described in some literature as a story about Byzantine-era generals trying to co-ordinate an attack, with possible traitors among the generals and/or their messengers. The point of this story is mutual agreement – agreement wins, disagreement looses. (There are thousands of papers on this subject.)
  (see L. Lamport, R. Shostak, M. Pease. *The Byzantine Generals Problem*, ACM Transactions on Programming Languages and Systems, Vol. 4, No. 3, July 1982, pages 382-401;
  http://research.microsoft.com/en-us/um/people/lamport/pubs/byz.pdf)

- Typical map to real world: Generals = processors, Messengers = data network communication

- Note: "Byzantine" is not synonymous with "bizarre". "Byzantine" has a precise meaning which deals with failure behavior that works against reaching agreement.

- In our SAFECOMP 2003 paper, we created concise practical definitions for designers:
  – Byzantine **fault**
    any fault that presents different symptoms to different observers
  – Byzantine **failure**
    the loss of a system agreement service due to a Byzantine fault
  (see K. Driscoll, B. Hall, H. Sivencrona, P. Zumsteg. Byzantine Fault Tolerance, from Theory to Reality, LNCS Computer Safety, Reliability, and Security, Volume 2788/2003, pp. 235-248, 2003;
  http://www.cs.indiana.edu/classes/p545-sjoh/read/Driscoll-Hall-Sivencrona-Xumsteg-03.pdf; or
  [better version] The Real Byzantine Generals, 23rd Digital Avionics System Conference (DASC), 2004)

- Any operational data link between redundant devices must exist for some type of agreement.
  Even asynchronous systems without voting need "equalization" to prevent divergence.

- It is nearly impossible to create a highly-dependable system without Byzantine Fault tolerance.

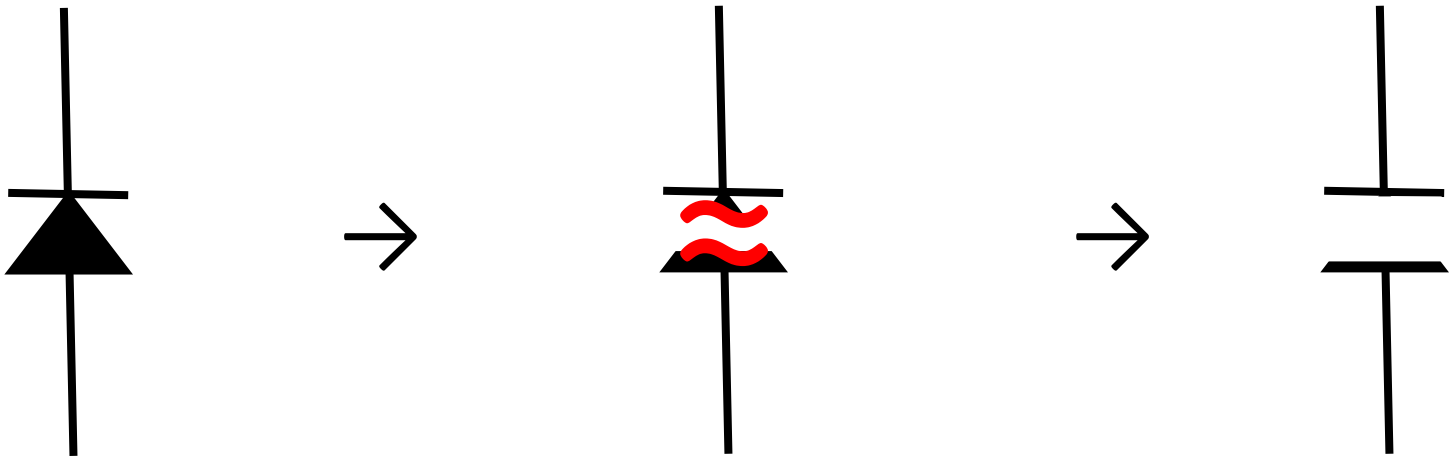
First Picture of a Byzantine Fault?

At 12:12 GMT 13 May 2008, a NASA Space Shuttle was loading hypergolic fuel for mission STS-124 when a 3-1 split of its four control computers occurred. Three seconds later, the split became 2-1-1. During troubleshooting, the remaining two computers disagreed (1-1-1-1 split). ***Complete system disagreement.*** But, none of the computers or their intercommunications were faulty! The ***single fault**** was in a box (MDM FA2) that sends messages to the 4 computers via a multi-drop data bus that is similar to the MIL STD 1553 data bus. This fault was a simple crack (fissure) through a diode in the data link interface.

Figure 1. Two views (90 degrees apart) of a fissure that appears to go through the silicon - Red arrows.

* the Byzantine Assassin

Transmogrification*

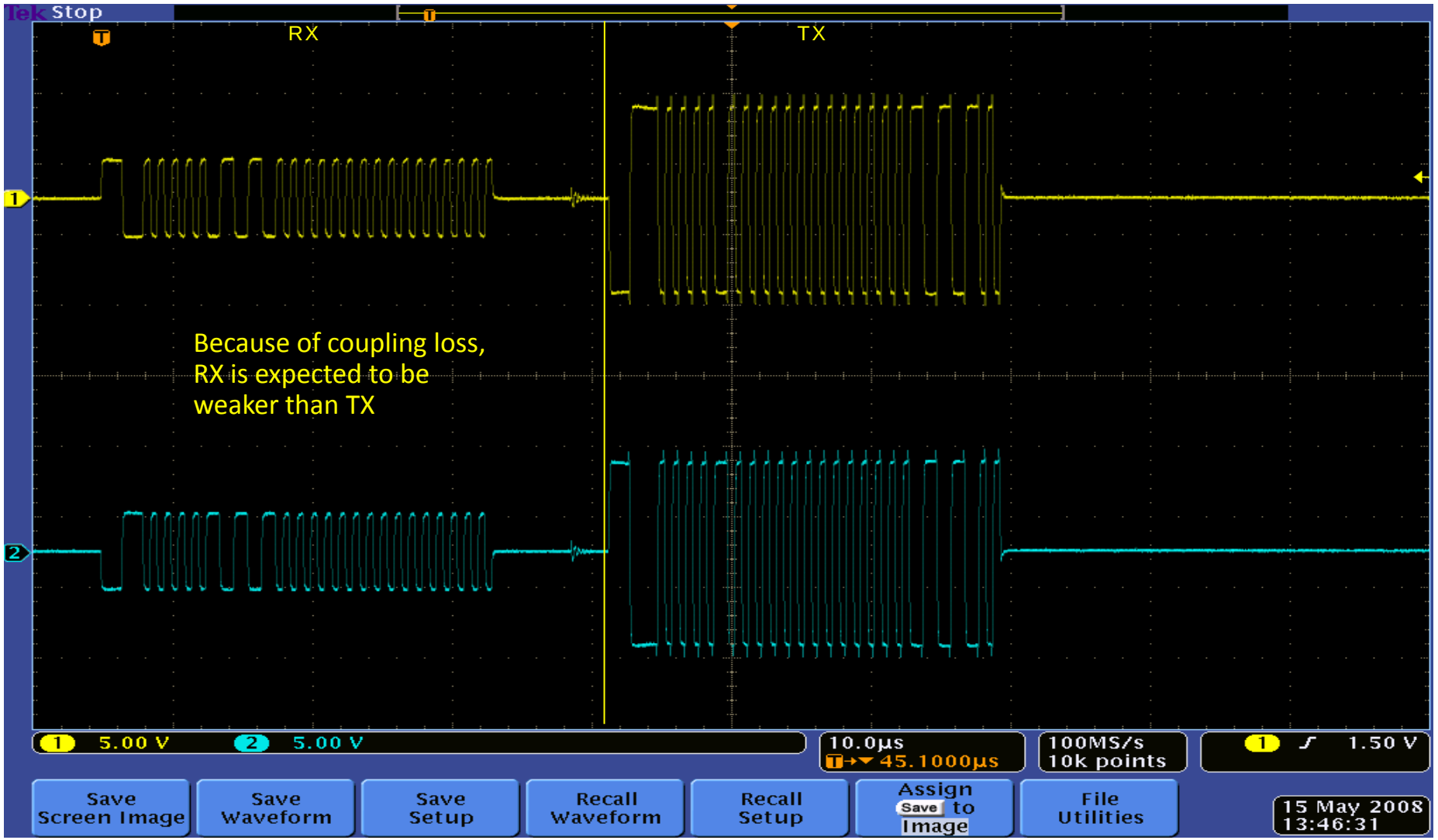


A diode … with a perpendicular crack … is really a capacitor

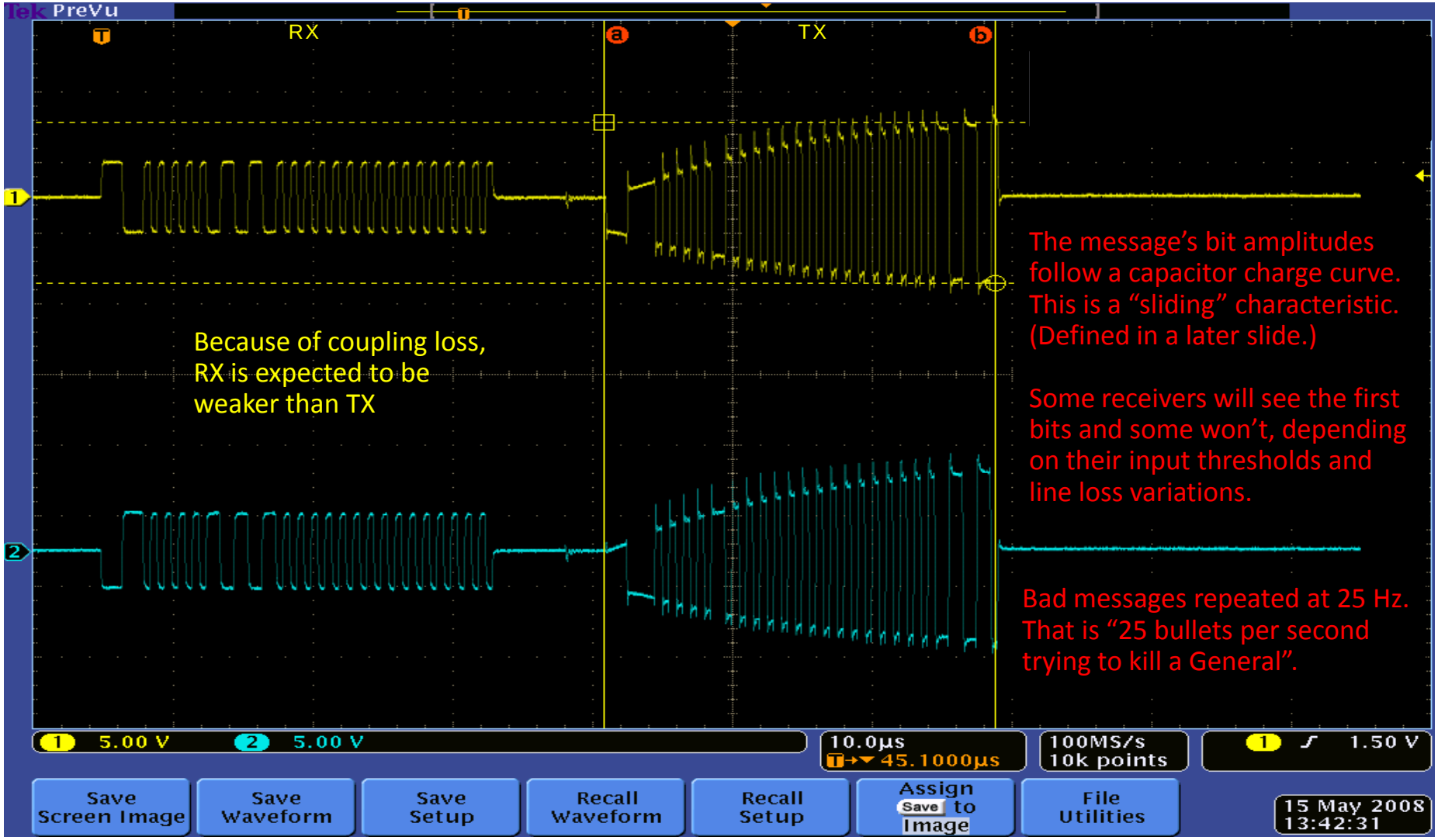
How many Failure Modes and Effects Analysis (FMEA) procedures ask what would happen if one electrical part (a diode) changed into another (a capacitor)? And, yet, the simplest of failures (a crack) caused this transmogrification. The literature includes other examples of capacitors becoming resistors, transistors becoming silicon controlled rectifiers (SCRs), amplifiers becoming oscillators, …

* Transmogrification definition: the act of changing into a different form or appearance (especially a fantastic or grotesque one), often as if by magic

Normal Messages (differential traces)



Faulty TX Message on the Right



Summary of STS−124 Observations

- A Byzantine Assassin can …
  - … be an Outsider (not a General nor one of the Generals' Messengers)
  - … be created by the simplest of faults (e.g. crack) in the simplest of parts (e.g. diode)
  - … convince "good guys" to kill (or ostracize) themselves
  - … cause as many corpses (or cliques) as there are entities to attack
    - ➢ Manual reversion to the shuttle's dissimilar backup may not have helped

      → ***Without Byzantine Fault tolerance, no amount of redundancy is enough***
- "Murder mechanisms" (e.g. vote-out reconfiguration, hybrid NMR) are inherently dangerous
  - A Byzantine Assassin can subvert the mechanism into being an accomplice for mass murder
  - Suicide is safer; that is one reason to use atomic self-checking pairs (e.g. Boeing 777 AIMS)
- Need to consider "sliding failures"
  - A part's behavior gradually changes
    - From in-specification to (slightly) out-of-specification, or vice versa
  - Higher probably of hitting a Byzantine region of behavior than one would expect
- Faults can convert one type of part into another
  - Diode → capacitor, capacitor → resister, transistor → SCR, input → output, amplifier → oscillator, analog circuit → digital circuit, digital circuit → analog circuit
  - Related phenomenon: a fault can create a part from "nothing" ("partogenesis" ☺)
    - Typically, due to the fault causing a large increase in some parasitic properties
    - Today's higher speed circuits are more susceptible to parasitic property changes
    - Emergent properties is another source of partogenesis
- FMEA teams should include:
  - Curmudgeons, skeptics, & "pathological thinkers" (to counterbalance designers, who are optimists)
  - Members of related/neighboring disciplines
  - Physicists (find other Feynmans)

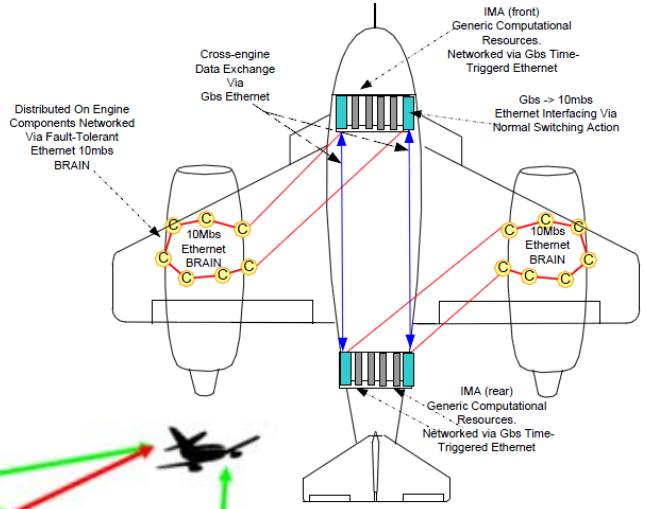
Arthur C. Clark's 1st law of prediction: "When a distinguished but elderly scientist states that something is possible, he is almost certainly right. When he states that something is impossible, he is very probably wrong."

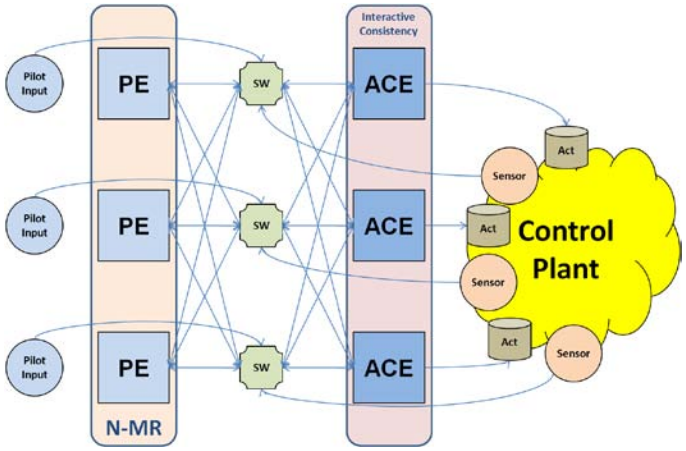
Communication and Distribution

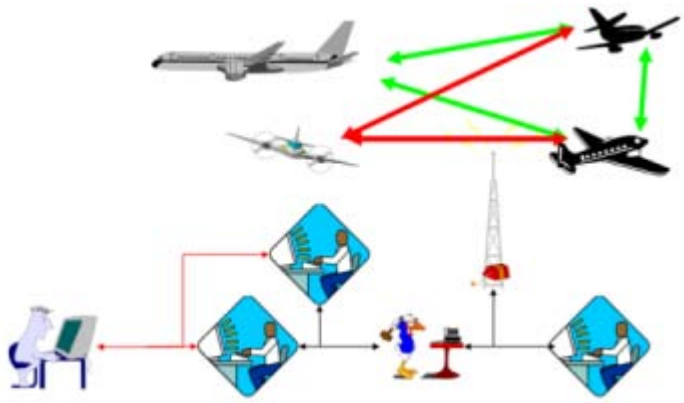
- Integrate modeling framework with architectural modeling standards – AADL
- Validate and refine modeling frameworks with real world examples
  - Asynchronous self-checking high integrity architecture – flight control
  - Time-triggered self-checking high-integrity architecture – Boeing 777 AIMS
  - Time-triggered simplex – e.g. MAC FADEC
  - Simplex-node voted-integrity architecture (Spider, MAFT)
- Integrate detailed formal analysis case study – TTEthernet
  - TTEthernet was designed with formal methods “in the loop”
    - Integrate existing proof’s into Evidential Tool Bus
    - Extend formal analysis
      - New services, e.g. group membership
  - Validate test case generation from formal models
    - Allow rapid “corner case” discovery
    - Link tests into formal evidential to bus framework

New Systems Decompositions and Functional Integrators

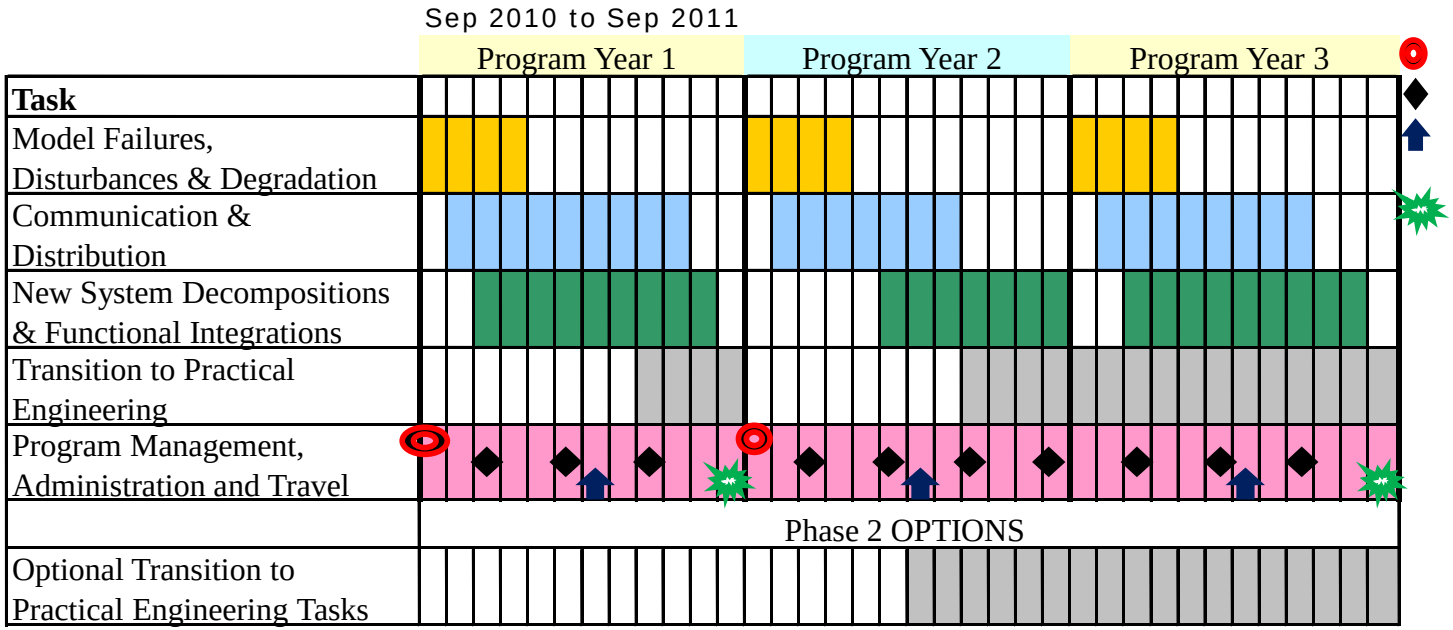
- Apply and evolve modeling and analysis framework for complex systems
  - Framework for mixed mode and hybrid systems
    - Analysis of example TT control system with reversionary asynchronous mode
    - Hybrid model, include plant abstraction
  - Heterogeneous network architectures
    - Mixed mode TT: TTEthernet, FlexRay, BRAIN
    - Asynchronous TT integration, AFDX Integration
  - Next Gen Systems
    - E.g. Airborne Traffic Situational Awareness







Schedule



Program Milestones:

Program Phase Kick-off

Interim Report

Avionics Safety Conference

Final report

Progress to Date

- Initial AADL modeling completed
  - Four example systems
    - SAFEbus – self-checking bus
    - TTP – simplex time-triggered protocol
    - SPIDER – voted Byzantine tolerant broadcast
    - BRAIN – qualified integrity forwarding (ring/mesh)
  - Some issues identified with AADL error model representation
    - We are bringing examples to AADL working group SAE AS-2C AADL
- Probabilistic modeling with PRISM
  - Modeled SPIDER broadcast with PRISM
    - Work indicated potential scalability problems
- EDICT Modeling
  - Explored several architectural mechanisms from example systems
    - Work has resulted in improvements to EDICT error mitigation representation and error propagation analysis strategies (work on going)
  - Explored “out-of-band” error propagation framework
    - Working on EDICT influencer modeling framework to capture environmentally induced faults

Backup Slides

Error Terminology &Taxonomy

- The AADL *Error* Annex is not a fault or failure annex.
  - But, you could fake it (for every case??) with terminology changes (i.e. replace "error event" with "fault" or "failure").
- Intent of annex is low level "bit flips", their propagation, and handling.
- Need to include
  - multiple levels
  - (semi-)semantic tunneling
    - Semantic-free pass-through
    - Classes of partial pass-through
- Need to add out-of band faults
  - Failure modes that aren't errors
  - Failures from the environment
  - Failures through the environment

A good taxonomy can act as an "examination of conscience".

Example of Layered Modeling

- TTP Bus
  - Very good example of “dependability Interaction”
    - Protocol components proved correct
      - Yet failed due to composition
  - Protocol “itself” acts a fault propagation mechanism
    - Membership Services has an amplified response to SoS faults
    - Protocol fault vulnerability also influenced by protocol state
      - E.g. bad initial Integration Frame vs. bad C-state once running
      - *To capture system failure modes, we need to consider protocol behavior*
  - Protocol also influenced by software
    - Life-sign interaction on start-up and membership
- TTP + Hub
  - Allows analysis of mitigation strategy at system level
  - Hub mitigation closely coupled to protocol but extends fault-tolerance
    - Babbling dual lane interface, SoS containment
  - Hub also dependent on protocol behavior
    - Parasitic sync, schedule detection
  - Hub fault containment also a function of Protocol modes
    - Start-up, limited protection, coupled to noise tolerance
    - E.g. Lost - SoS only, prioritized channel arbitration
    - Sync mode, SoS and slot enforcement

Error Terminology &Taxonomy Definition

The naming system is shown here using the POSIX extended regular expression representation.

(([es]_?[ip]?)|p)_([bo]|([tv][as][dnq])(_m_[^_]+)?(_nc)?((_remission|_repair)?_rate)?

The first character (e, p, or s) is used to differentiate among error propagation, error state, and error event. These are the three main declared items within a fault model per the AADL Error Annex.

- e ⇒ error *e*vent
- p ⇒ error *p*ropagation
- s ⇒ error *s*tate

The next set of characters, from the first underscore until the second underscore, are used to denote the error manifestation. These are outlined below.

- a ⇒ error will manifest *A*symmetrically among receivers, i.e. Byzantine error
- b ⇒ semantic-free *B*abble that leads to denial of service
- d ⇒ error is *D*etectable by in-line acceptance testing
- i ⇒ error will manifest *I*ntermittently, represent transient error behavior
- n ⇒ error is *N*ot detectable by in-line acceptance test
- o ⇒ *O*mission, fail-stop
- p ⇒ error will manifest *P*ersistently i.e. that it is a permanent behavior
- q ⇒ data that has been flagged as *Q*uestionable, untrusted
- s ⇒ error will manifest *S*ymmetrically, presenting the same value to all receivers
- t ⇒ a data *T*emporal error
- v ⇒ a data *V*alue error

The next two optional items deal which errors which are meaningful only when including some context outside of the local component. These are:

- m ⇒ *M*eaning of error is outside of the local context
- nc ⇒ data is *N*ot *C*onsistent with another copy/flow

The syntax for the *M*eaning error is ”m” immediately followed by the name of the component that caused this error.

The last three items are used only for setting the probabilities or rates of events.

- rate ⇒ error event rate
- remission rate ⇒ self-healing rate (for intermittent and transient errors)
- repair rate ⇒ repair rate