Architecture-Level Security Concerns in a Safety Critical System

Sam Procter

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Copyright / Distribution Information

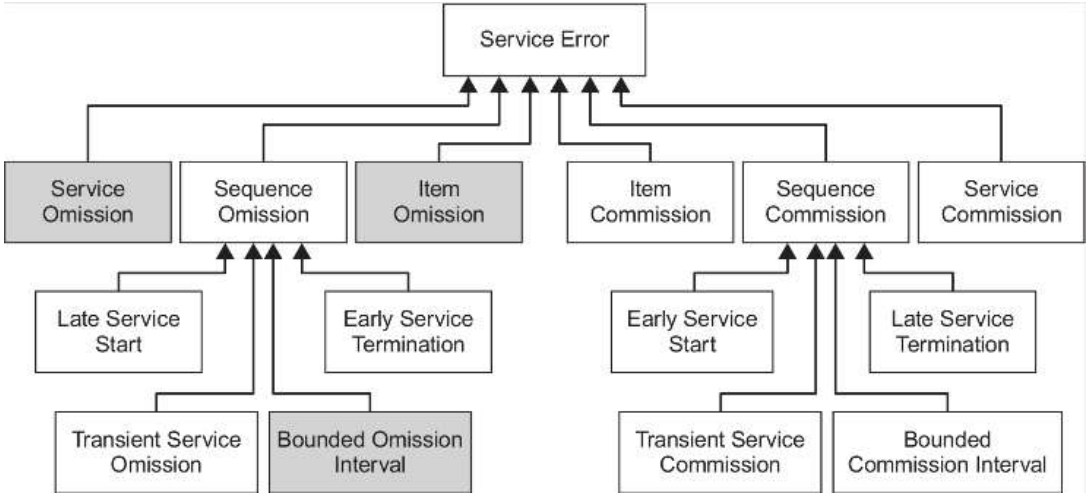
This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM18-1288

Aside: This talk vs. My paper

Paper topic: An *operationalized* taxonomy of system errors

Not covering this topic directly in this talk, but I'm happy to answer questions about it



Agenda

0. AADL Primer

1. Safety in AADL
2. Security in AADL
3. Safety + Security

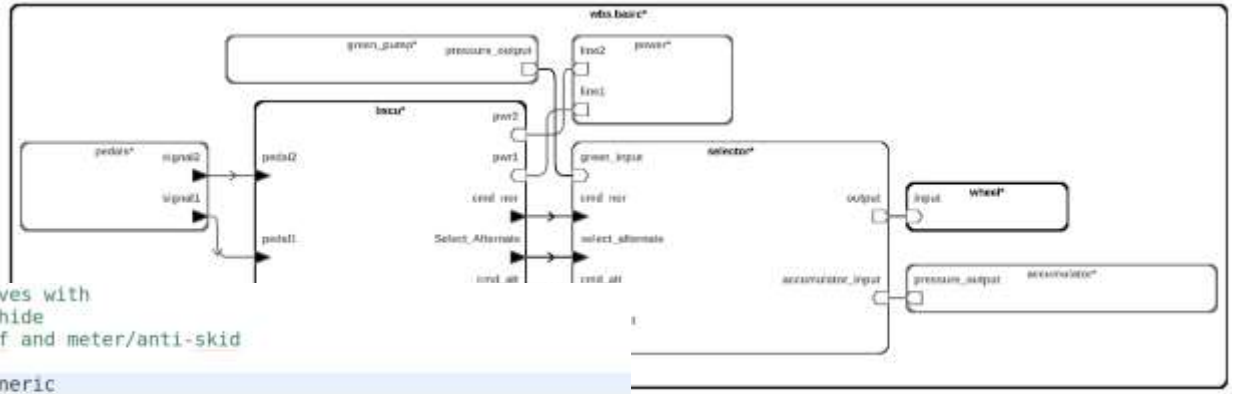
AADL: The language used for this work

AADL focuses on interaction between the three elements of a software-reliant mission and safety-critical systems

What does AADL actually look like?

Semi-formal semantics

Only architectural elements




```
79 --  Basic/naive version that abstracts all the valves with
80 --  a selector subsystem. This selector subsystem hide
81 --  the physical logic behind the selector, shutoff and meter/anti-skid
82 --  valves.
83     system implementation wbs.basic extends wbs.generic
84 subcomponents
85     bscu : refined to system impl::bscu::bscu.basic;
86     --  The selector subsystem
87     selector   : refined to system impl::valves::selector_basic{Classifier_Substitution_Rule => Type_Exte
88     wheel      : refined to system impl::wheel::wheel_one_input.i{Classifier_Substitution_Rule => Type_Ex
89 connections
90     blue_to_selector    : bus access blue_pump.pressure_output  <-> selector.blue_input;
91     green_to_selector   : bus access green_pump.pressure_output <-> selector.green_input;
92
93     bscu_sel_to_selector : port bscu.Select_Alternate -> selector.select_alternate;
94     bscu_cmdnor_to_selector : port bscu.cmd_nor -> selector.cmd_nor;
95     bscu_cmdalt_to_selector : port bscu.cmd_alt -> selector.cmd_alt;
96
97     selector_to_wheel   : bus access selector.output <-> wheel.input;
98 end wbs.basic;
```

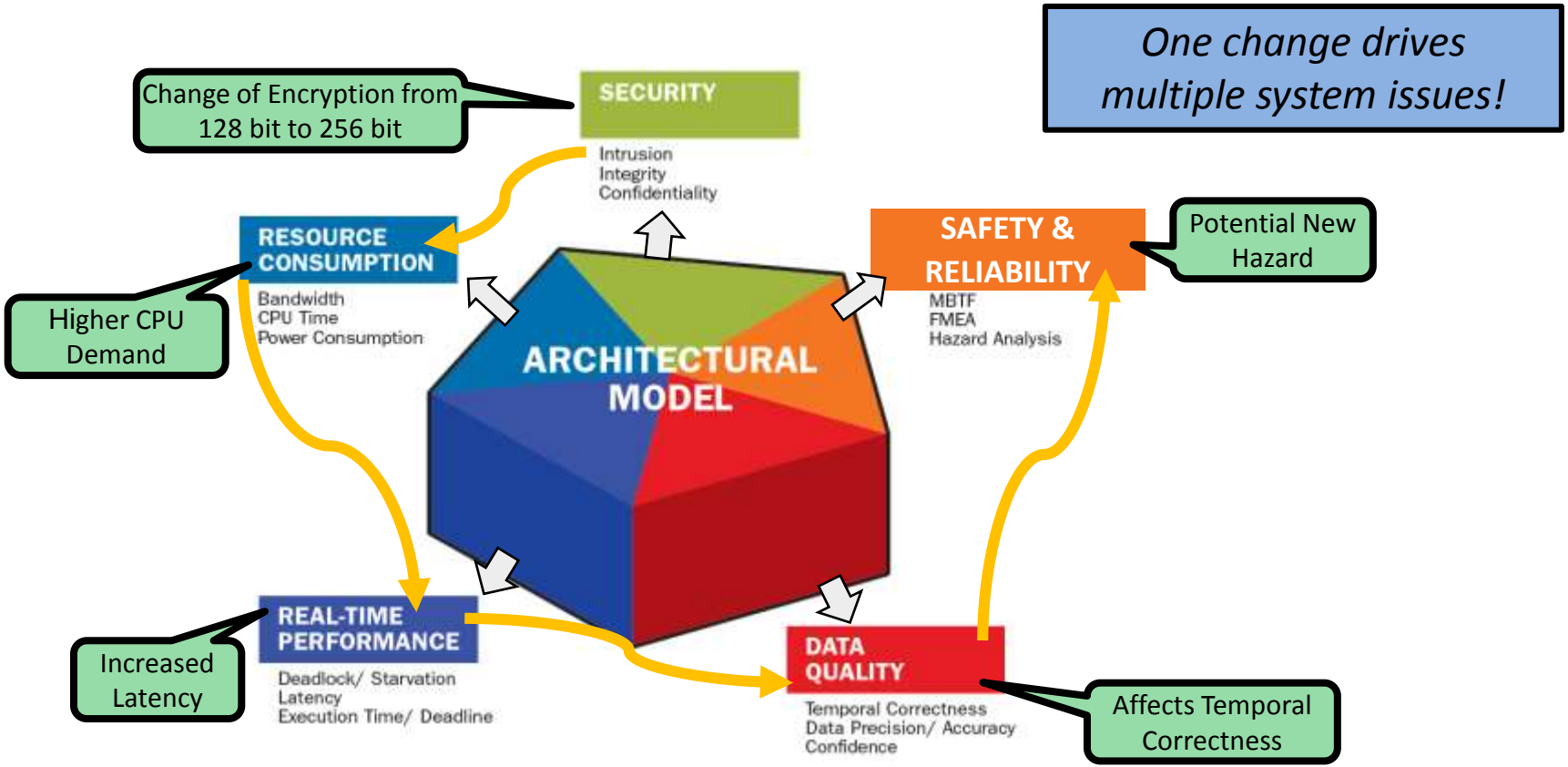
Annexes add functionality:

- Error Modeling
- Behavior
- Code Generation

AADL excels at analyzing component-based systems by

- integrating annotated components
- running system-level analyses

The benefit of a “Single Source of Truth”



Agenda

0. AADL Primer

1. Safety in AADL

   1. Background
   2. ALISA + EMV2
   3. Why generate reports?

2. Security in AADL

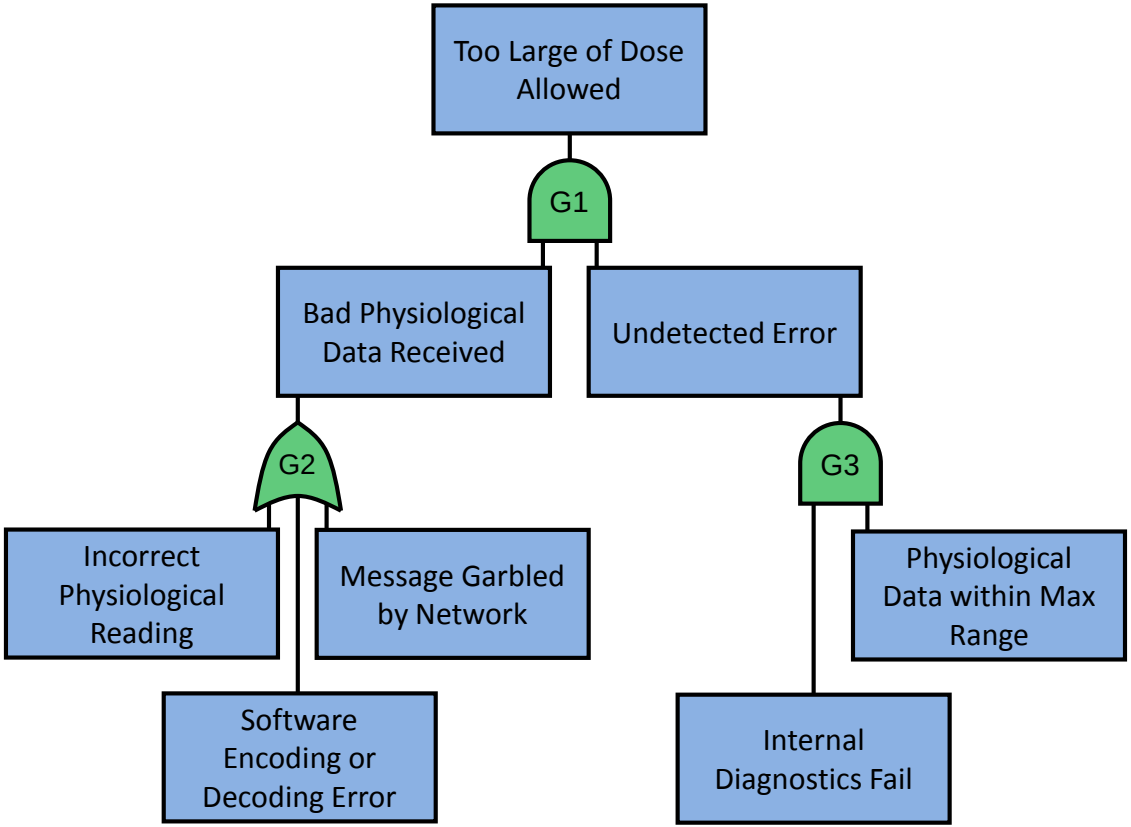
3. Safety + Security

Safety Background: Fault Tree Analysis

Bell Labs, 1962

Looks for contributory causes to undesired events

Doesn't really have a notion of "component" or use system structure



Safety Background: Failure Modes and Effects Analysis

FMEA: US Military, 1949

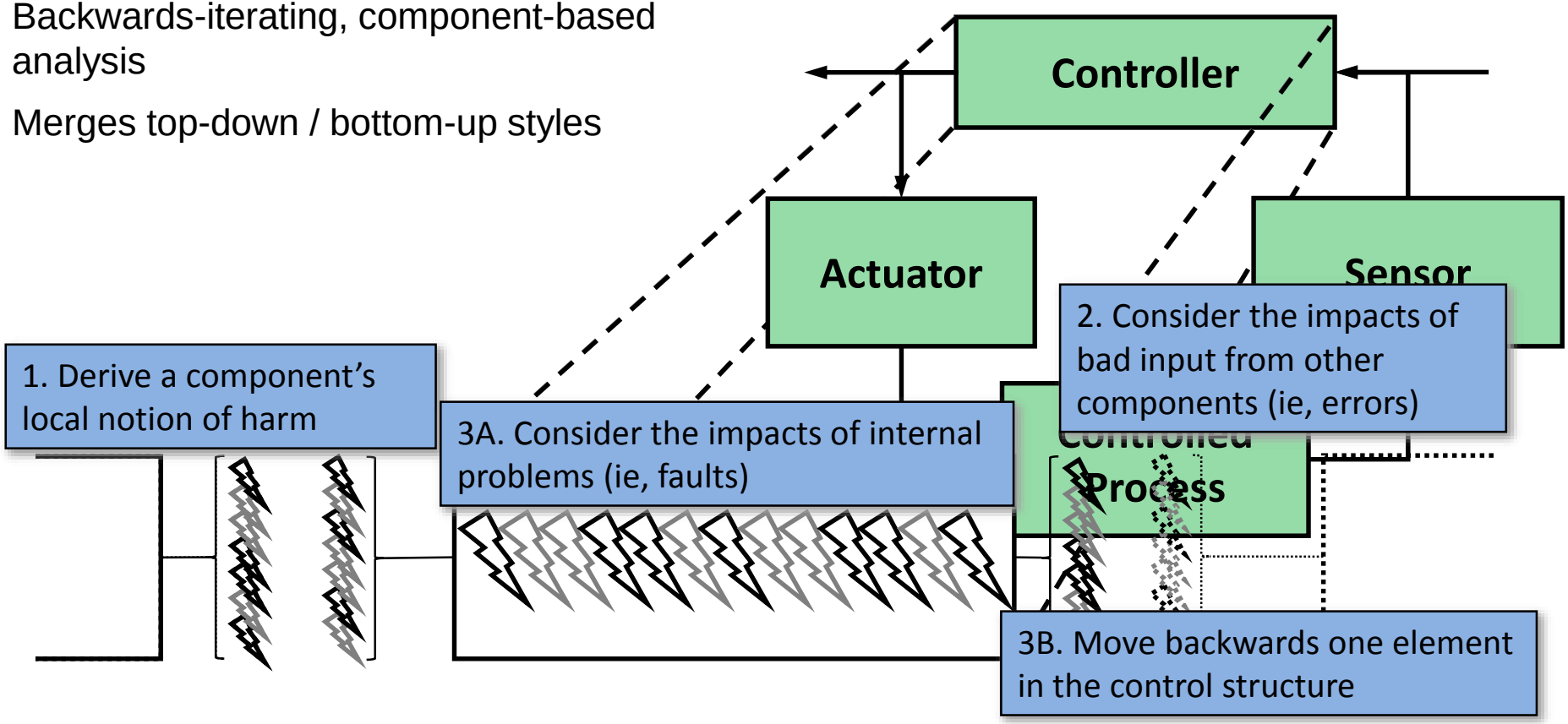
- Analyses impacts of individual components
- Doesn't clearly address component-interaction problems

System: PCA Interlock Scenario			Subsystem: Pulse Oximeter Device				Mode/Phase: Execution			
Function	**Failure Mode**	**Fail Rate**	**Causal Factors**	**Effect**	**System Effect**	**Detected by**	**Current Control**	**Hazard**	**Risk**	**Rec. Action**
Provide SpO_2	Fails to Provide	N/A	Network or dev. Failure	No SpO_2 data	Unknown patient state	App		Potential OD	3D	Default to KVO
	Provides late	N/A	Network slowness	No SpO_2 data	Unknown patient state	App		Potential OD	3C	Default to KVO
	Provides wrong	N/A	Device error	SpO_2 wrong	Wrong patient state	None		Potential OD	1E	Dev. should report data quality
Analyst: Sam Procter			Date: September 26, 2016				Page 3/14			

Safety in AADL: Research Background

Backwards-iterating, component-based analysis

Merges top-down / bottom-up styles



How do you incrementally assure a system?

Start early – link requirements to:

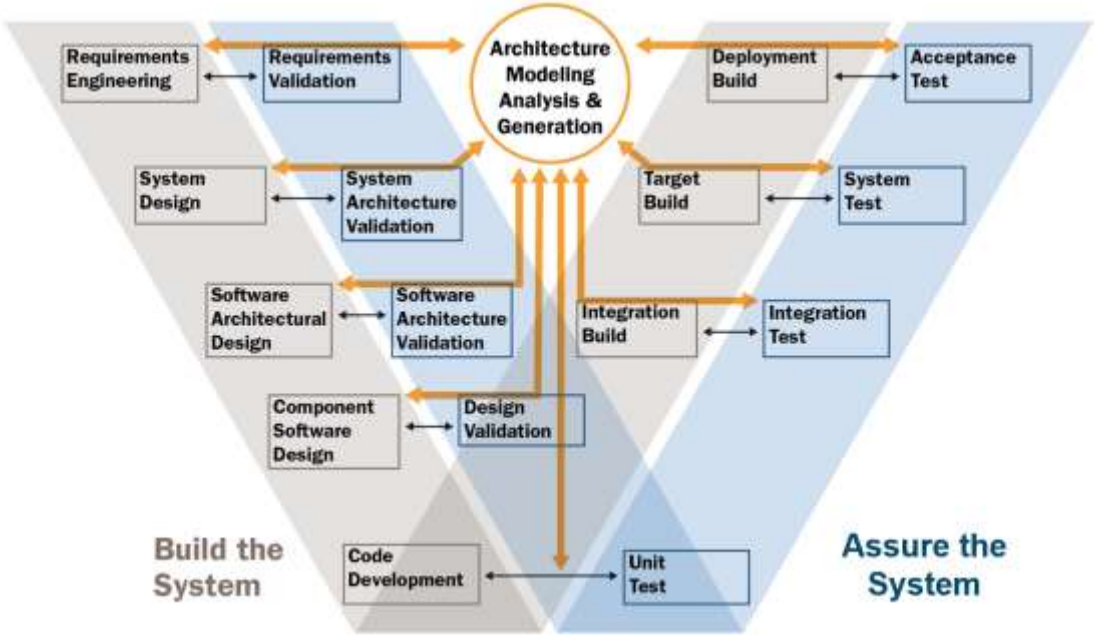
- Each other
- Architectural components

Document:

- Goals, stakeholders, etc.
- Verification plans

Generate:

- Coverage reports
- Hazard analyses



ALISA Example

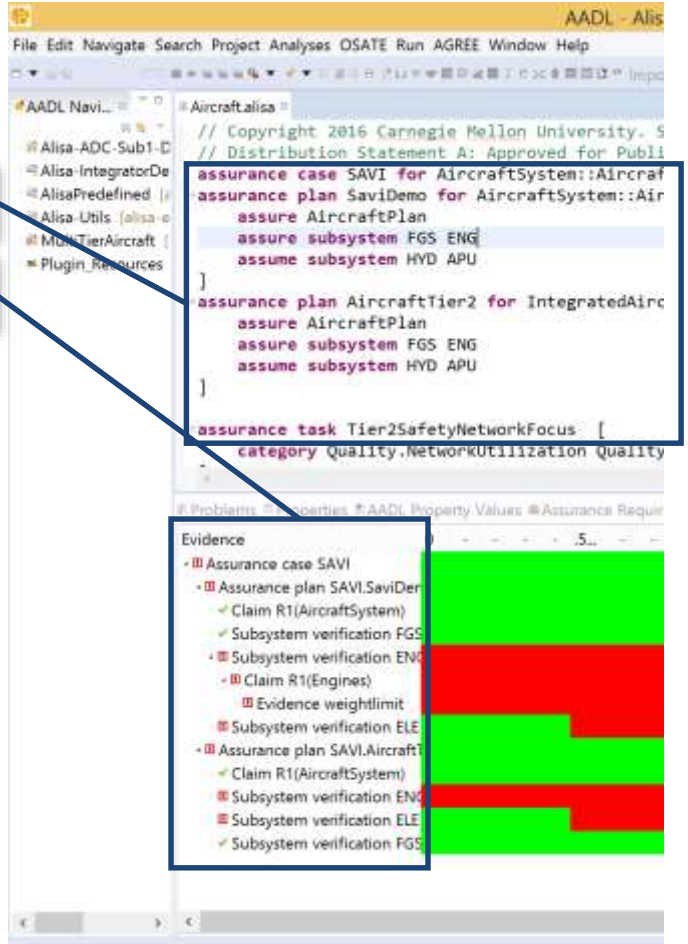
Hierarchical assurance plan

JUnit-Style evidence evaluation

Human-readable description

Goal link

```
system requirements caccreqs:"CACC"
for caccIntegration::cacc_rt.devices
use constants caccConstants
[
    val MaximumSpeed = 120.0 mph
    val SpeedLimit = 70.0 mph
    compute CurrentSpeed: caccProperties::Velocity_Type

    requirement speed_R1 : "throttle setting cannot exceed the maximum speed"
    [
        description this " shall result in a speed not exceeding " MaximumSpeed
        rationale "fly by wire may introduce an electrical error
beyond the physical throttle setting"
        value predicate CurrentSpeed < MaximumSpeed
        mitigates "Invalid data sent by the speedometer"
        issues "need to recognize that physical subsystems
can present issues for a digital system"
        see goal caccStakeholderGoals.g1
        category quality.safety
        uncertainty[
            volatility 2
            impact 3
```



EMV2: Contracts for Error Behavior

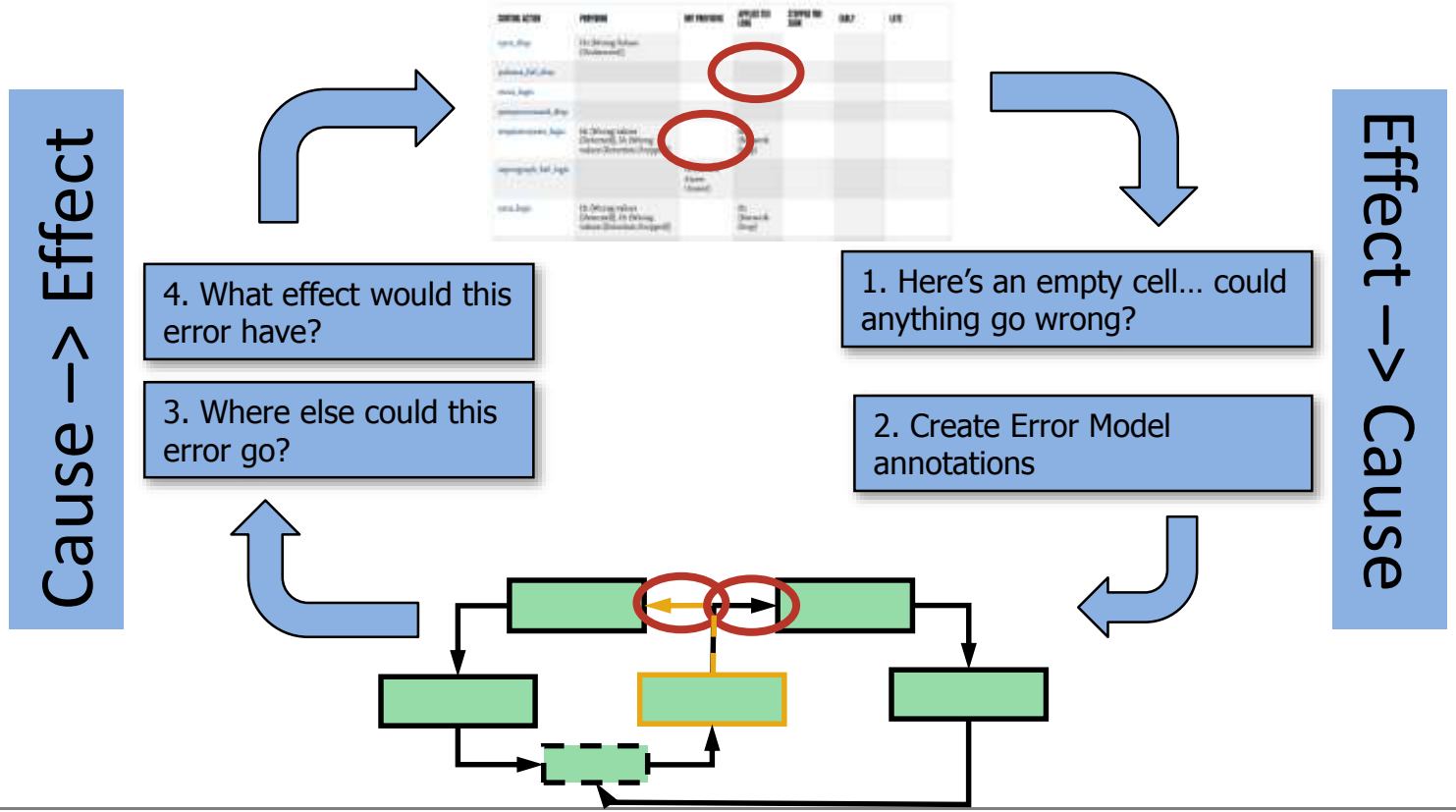
```
package PCA_Shutoff_Logic
public
with PCA_Shutoff_Types, PCA_Shutoff_Properties, MAP_Properties;

   process ICEpcaShutoffProcess
   features
      SpO2 : in event data port PCA_Shutoff_Types::SpO2;
      CommandPumpNormal : out event data port PCA_Shutoff_Types::PumpNormalCommand;
   properties
      MAP_Properties::Component_Type => logic;
   annex EMV2 {**
      use types PCA_Shutoff_Errors;
      error propagations
         SpO2 : in propagation {SpO2ValueHigh};
         CommandPumpNormal : out propagation {InadvertentPumpNormally};
         flows
            HighSpO2LeadsToOD : error path SpO2{SpO2ValueHigh} -> CommandPumpNormal{InadvertentPumpNormally};
      end propagations;
   **};
   end ICEpcaShutoffProcess;

   -- Process implementation redacted
end PCA_Shutoff_Logic;
```

Expected incoming errors

Expected outgoing errors

Mapping between inputs and outputs

Interaction between report generation and error propagation



Agenda

0. AADL Primer
1. Safety in AADL
2. **Security in AADL**
   1. Background
   2. AADL & MILS
   3. Security Policy Specification and Enforcement
3. Safety + Security

Security in AADL: Research Background

1970s: Multi-level security

- Bell-LaPadula (Confidentiality)
- Biba (Integrity)

2000s: Multiple Independent Levels of Security

- Local Policy Assurance
- Integrating Policy Assurance
- Individual Resource Separation Assurance
- Integration Resource-Sharing Assurance

MILS enforces *NEAT* properties:

- **N**on-bypassable
- **E**valuatable
- **A**lways Invoked
- **T**amperproof

AADL in large-scale formal methods: SMACCM & D-MILS

D-MILS

- Extension of MILS to networked systems
- Customized subset of AADL

image: d-mils.org

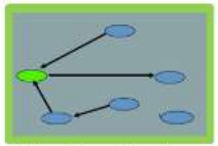
SMACCM

- “Unhackable” UAVs
- AGREE / Resolute

image: loonwerks.com

AADL Support for MILS

Functional Mission System Architecture



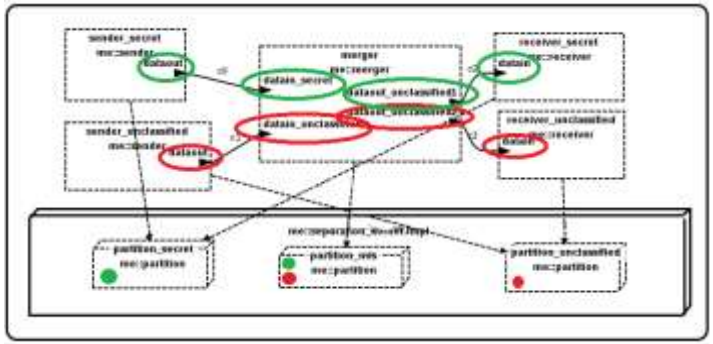
Mission System Security Policy

Security policy vulnerabilities: Analyze Information Flows

Examples: Verify secrets stay secret, and Sensors can't send commands

Security enforcement vulnerabilities: Analyze Deployment Mechanisms

Example: Hi and low-security channels shouldn't coexist on unpartitioned hardware

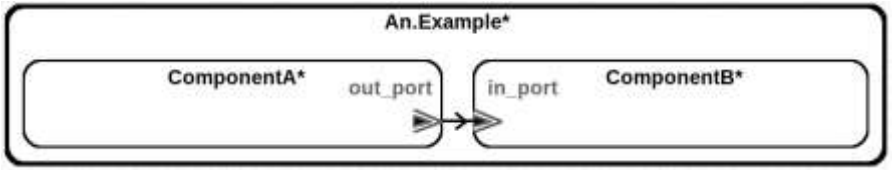


Research Connection:

Apply *Multiple Independent Levels of Security* (MILS) framework (confidentiality) to system security (integrity)

Partitioning code sample

```
1  package Sandbox
2  public
3      with SecurityProps;
4
5      system An
6      end An;
7
8      system implementation An.Example
9          subcomponents
10             ComponentA: system A;
11             ComponentB: system B;
12         connections
13             conn: port ComponentA.out_port -> ComponentB.in_port;
14         properties
15             SecurityProps::PolicyElements => SecurityProps::BibaPolicy;
16     end An.Example;
17
18     system A
19         features
20             out_port: out event data port;
21         properties
22             SecurityProps::level => high;
23     end A;
24
25     system B
26         features
27             in_port: in event data port;
28         properties
29             SecurityProps::level => low;
30     end B;
31
32
33 end Sandbox;
```



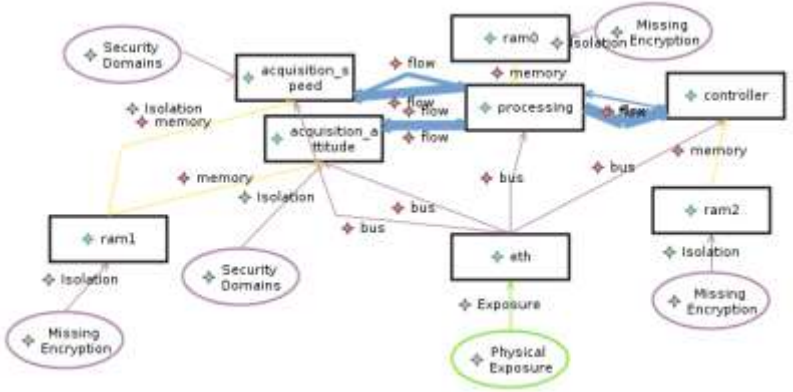
Policy specification

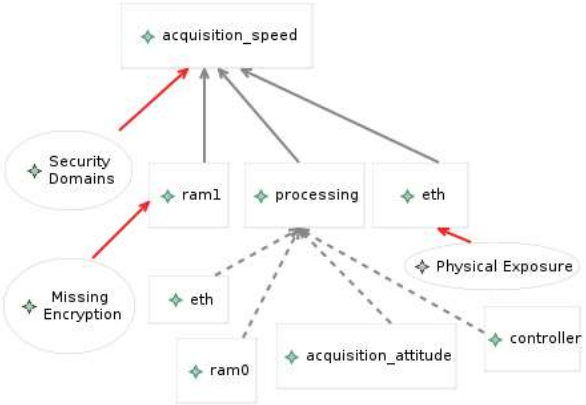
Security levels and / or partition

Conflict if we switch to Bell-LaPadula!

Security Analysis Techniques and Tools

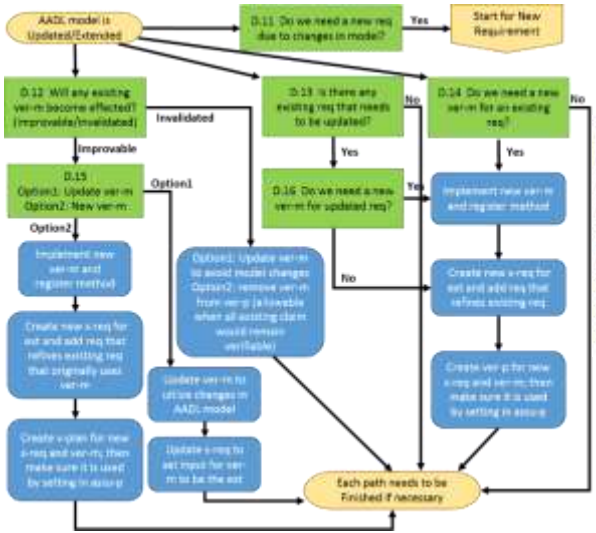
0. Consistency in security policy specification & enforcement
1. Model-Based Attack Impact Analysis (AIA) tool
2. Model-Based Attack Tree Analysis (ATA) tool
3. Generation of security configuration files
   - Model-based auto-configuration of certified kernel (seL4/CAmkES) security policy





Using Security Assurance Techniques and Tools

1. Specify security policy as verifiable requirements
2. Formalize verification activities
3. Automate execution of verification plans



MILS-R0: Components sharing a bus should have the same security level.
MILS-R1: Inter-communicating components should have the same security level.
MILS-R2: Processes with different security levels use isolated memory regions.
MILS-R3: Components associated with identical processing resources share the same security level.
MILS-R4: Threads inside the same process share the same security levels.
CWE-131 Incorrect calculation of buffer size.
CWE-311 Missing encryption of sensitive data.
CWE-805 Buffer Access with Incorrect Length Value.

Extension to Architecture-Led Incremental System Assurance (ALISA) workbench

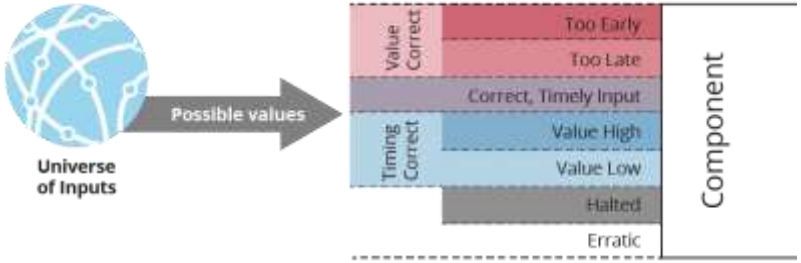
Agenda

0. AADL Primer
1. Safety in AADL
2. Security in AADL
3. **Safety + Security**
   1. Effects focus
   2. Code generation
   3. Slicing & Data-Flow

Modeling Security Requirements in the Context of Safety

Approach: Use effects-focused analysis and tooling

- When are various techniques appropriate?
  - Biba model (integrity)
  - Bell–LaPadula (confidentiality)
- What “building blocks” should be used?
  - examples: encryption, partitioning, checksums
- How should requirements be verified?



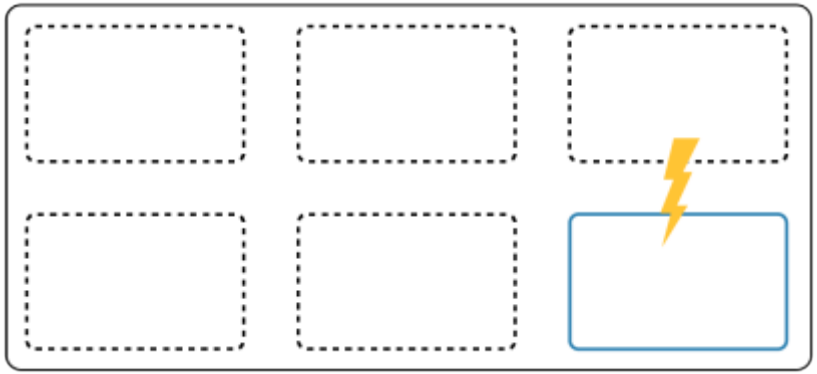
Measurement: Proposed user study (in FY 20) to measure qualities of design and analysis guidance

- Objective qualities
  - Number of issues found / avoided
  - Time required
- Subjective qualities
  - Quality of issues found / avoided
  - Complexity

Using Theory to Guide Tool Development

Approach: Use fault-injection tooling

- Fault-injection pairs naturally with an effects focus
- Collaborators are building a large simulation and verification environment to enable this testing

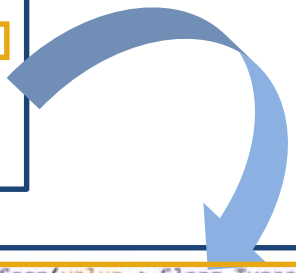


Measurement:

- Current AADL can describe component behavior in the presence of errors
- This project will let us verify those descriptions

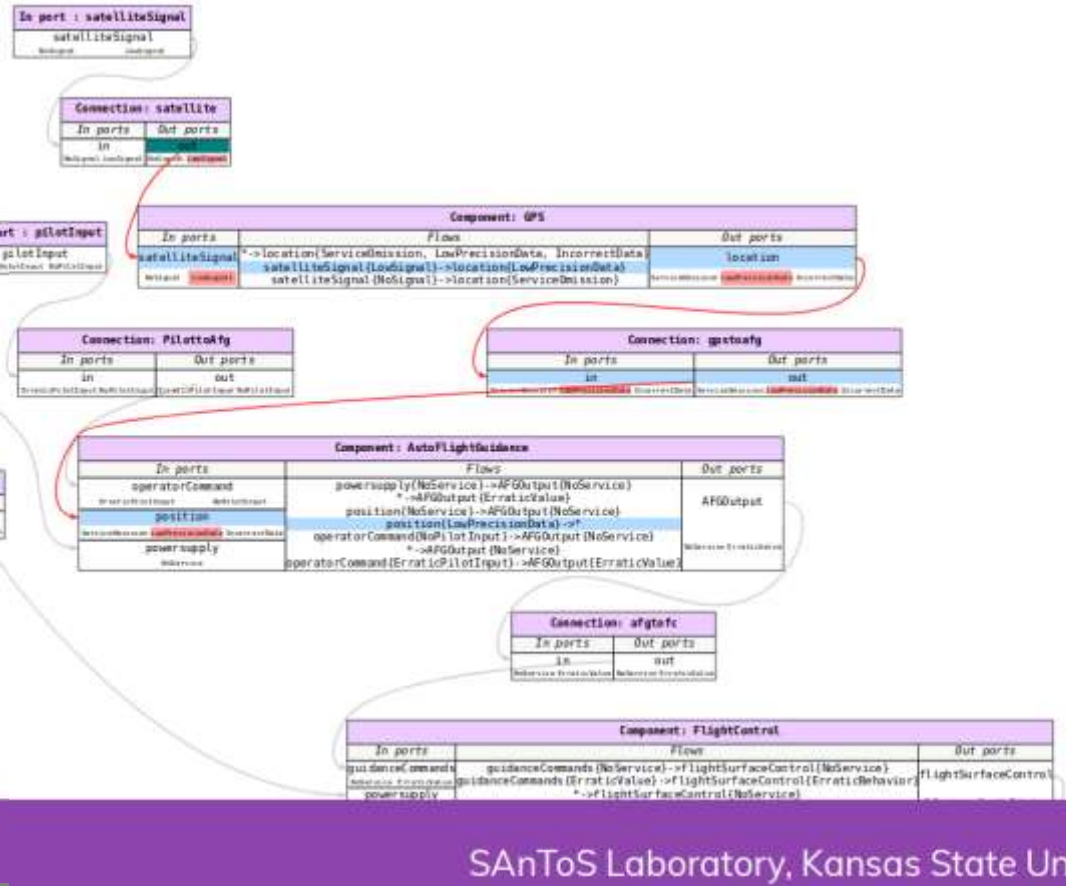
Code Auto-generated from AADL

```
thread Patient_Bolus_Checker
  features
    Minimum_Time_Between_Bolus: in data port ICE_Types::Minute;
    Patient_Button_Request: in event port;
    Patient_Request_Not_Too_Soon: out event port;
    Patient_Request_Too_Soon: out event port;
end Patient_Bolus_Checker;
```



```
def sendPatient_Request_Not_Too_Soon(value : Slang_Types.Empty) : Unit = {
  Art.putValue(Patient_Request_Not_Too_Soon_Id, Slang_Types.Empty_Payload(value))
}

def sendPatient_Request_Too_Soon(value : Slang_Types.Empty) : Unit = {
  Art.putValue(Patient_Request_Too_Soon_Id, Slang_Types.Empty_Payload(value))
}

def getMinimum_Time_Between_Bolus() : ICE_Types.Minute = {
  val ICE_Types.Minute_Payload(value) = Art.getValue(Minimum_Time_Between_Bolus_Id)
  return value;
}
```

Looking forward: Data-Flow Analysis



What do all these analyses have in common?

- The use the “data flow” view of a system

Colleagues at K-State (Hariharan Thiagarajan, John Hatcliff, Robby) are bringing data-flow and slicing to AADL models / generated simulation code.

We’re working on integrating this into our tool’s standard distribution

SAnToS Laboratory, Kansas State University

Architecture-Level Security Concerns in a Safety Critical System

Sam Procter

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Questions: Modeling Strategy

We don't model users – how do we model access control?

- Data types

We don't model state – how do we model protocols?

- Virtual buses

Larger question: *How* should security-related concepts be modeled?

- Should adding new concepts be a last resort?
  - This can give a nice, compact language
- … Or should they be added to avoid "hacks?"
  - This can make the language more readable

Related: *When* should security-related concepts be modeled?