



# Building Safe and Secure Systems with AADL

Software Engineering Institute  
Carnegie Mellon University  
Pittsburgh, PA 15213

Julien Delange  
AADL Meeting February 15



## Report Documentation Page

*Form Approved  
OMB No. 0704-0188*

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE <b>15 FEB 2015</b>	2. REPORT TYPE <b>N/A</b>	3. DATES COVERED	
4. TITLE AND SUBTITLE <b>Building Safe and Secure Systems with AADL</b>		5a. CONTRACT NUMBER	
		5b. GRANT NUMBER	
		5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) <b>Delange /Julien</b>		5d. PROJECT NUMBER	
		5e. TASK NUMBER	
		5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) <b>Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213</b>		8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)	
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT <b>Approved for public release, distribution unlimited.</b>			
13. SUPPLEMENTARY NOTES <b>The original document contains color images.</b>			
14. ABSTRACT			
15. SUBJECT TERMS			
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT
a. REPORT <b>unclassified</b>	b. ABSTRACT <b>unclassified</b>	c. THIS PAGE <b>unclassified</b>	
			19a. NAME OF RESPONSIBLE PERSON

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

This material has been approved for public release and unlimited distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at [permission@sei.cmu.edu](mailto:permission@sei.cmu.edu).

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM-0002054



# Agenda

**Introduction to AADL**

**AADL modeling patterns for safety and security**

**AADL validation tools dedicated to security and safety**

**Demonstration**



# Agenda

**Introduction to AADL**

**AADL modeling patterns for safety and security**

**AADL validation tools dedicated to security and safety**

**Demonstration**



# Introduction

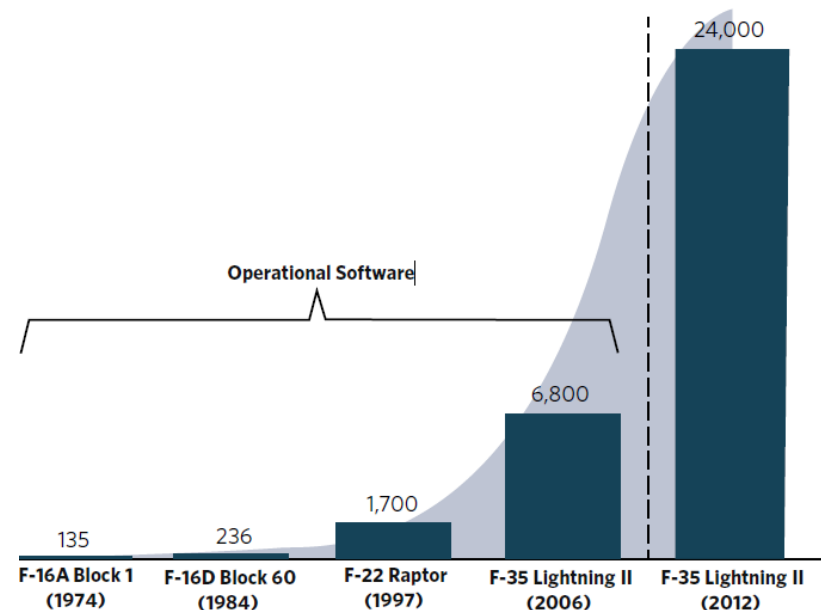
**Systems are becoming extremely software-reliant**

## Need to verify and validate requirements

- Requirements errors propagate through design
- Need to verify/validate requirements

## Major integration and coding issues

- Incur massive re-engineering rework
- Could be removed by early analysis



# Architecture Analysis and Design Language

## Model-Based Engineering with AADL

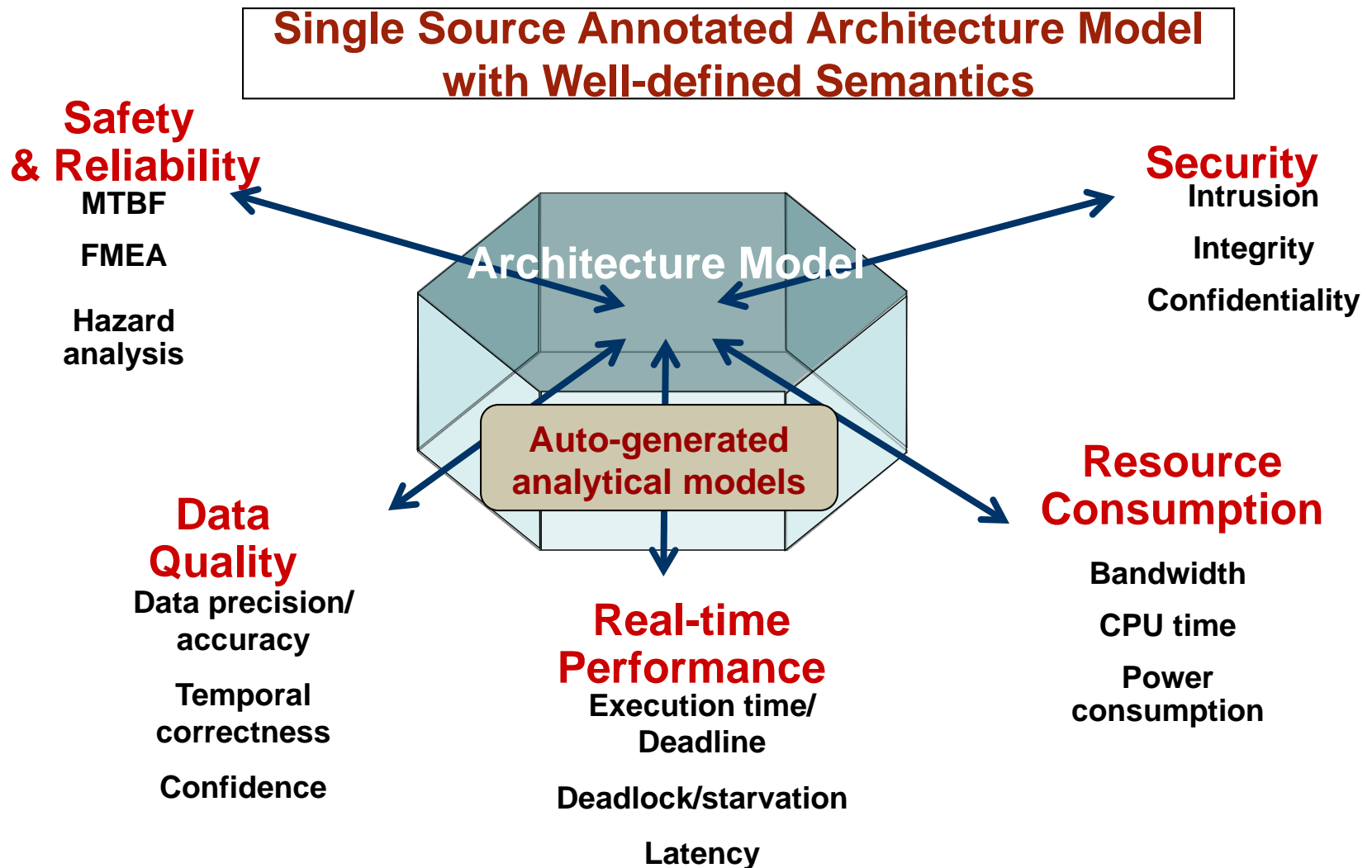
- Architecture Language Description standardized by SAE
- Description of Systems and Software Concerns
- Precise & unambiguous semantics
- Textual and Graphical Representation

## Support for Model Analysis

- Verify system requirements (i.e. latency, safety)
- Check model integration before producing the implementation

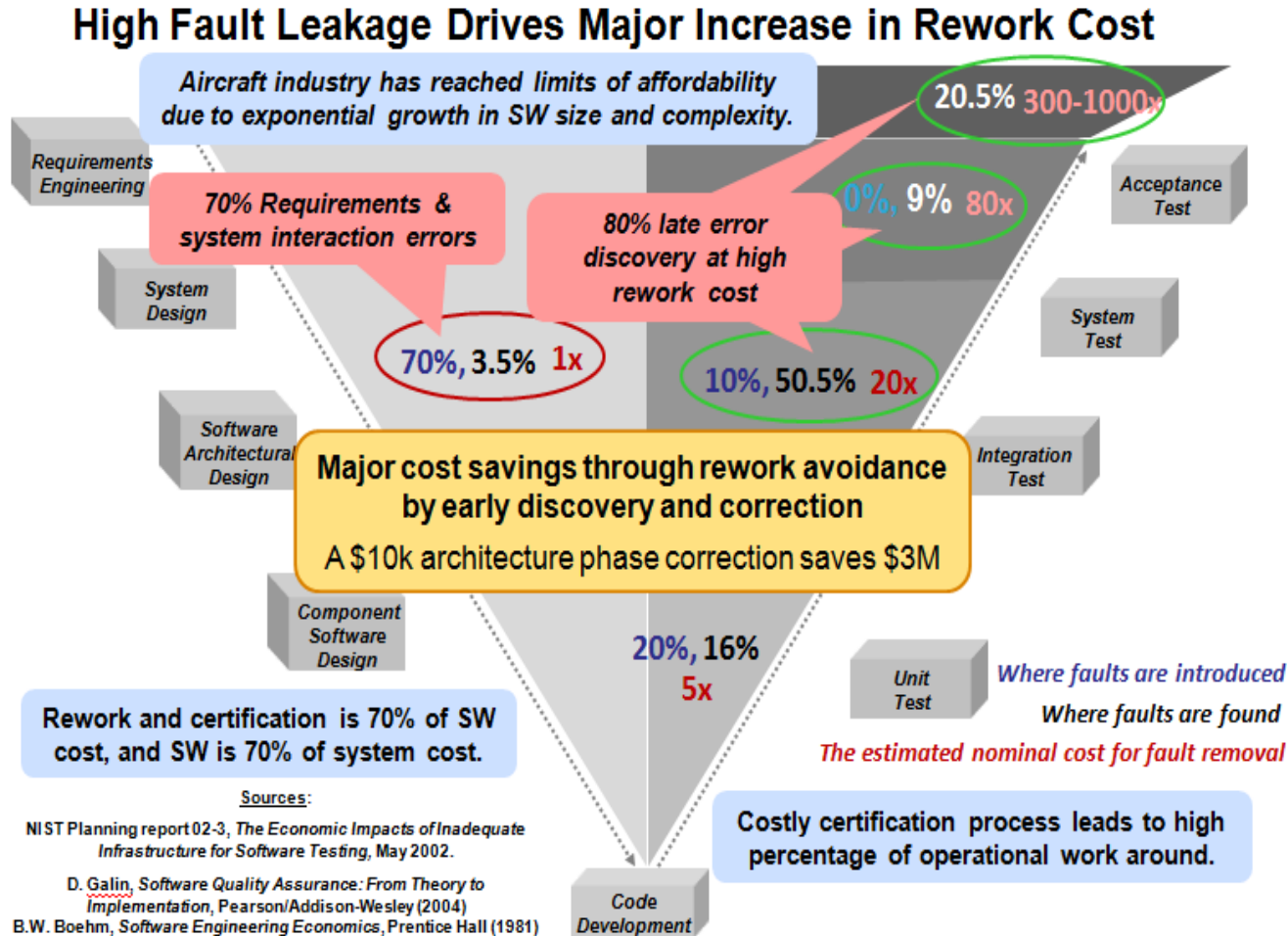


# AADL Model-Based Technology Overview





# Understanding Actual Software Issues



# Use of AADL in Development Process

## Software and Component Design

Define components requirements & interfaces

Early verification validation of components integration

## Code Development

Auto-Generate Code (AADL, Simulink, SCADE)

Avoid traditional coding errors

Ensure correct translation of requirements

## Unit & Integration Test

Automatic generation of tests from models

Reduce tests as system was validated earlier



# Agenda

Introduction to AADL

**AADL modeling patterns for safety and security**

AADL validation tools dedicated to security and safety

Demonstration



# Security Specifications

## Leverage AADL properties for security level specification

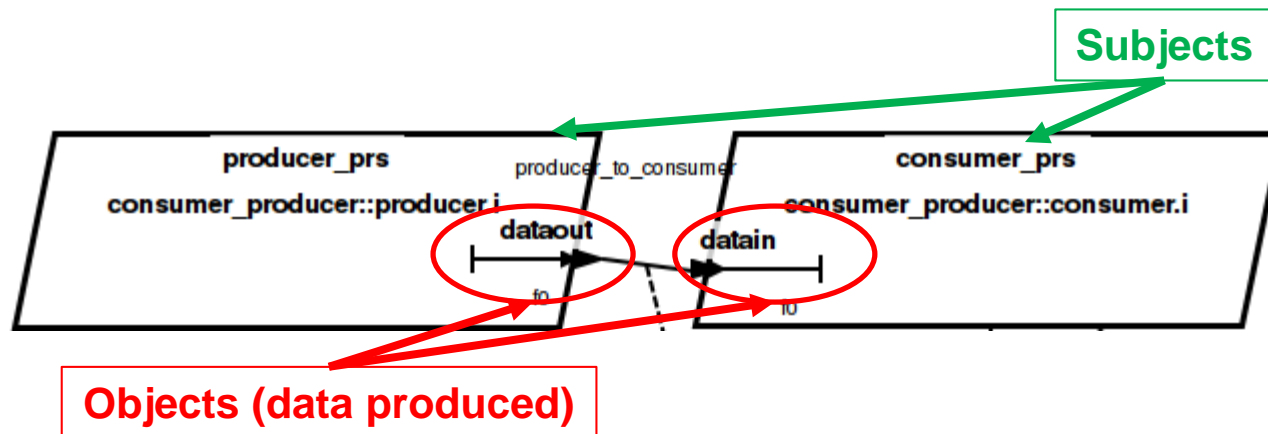
Define security-specific values

Associate them with components and interfaces

## Direct mapping to MILS Security Level concepts

MILS subjects to AADL runtime components

MILS objects to AADL interfaces



# Partitioning Policy (as in ARINC653 or MILS)

## Partitions content and attributes

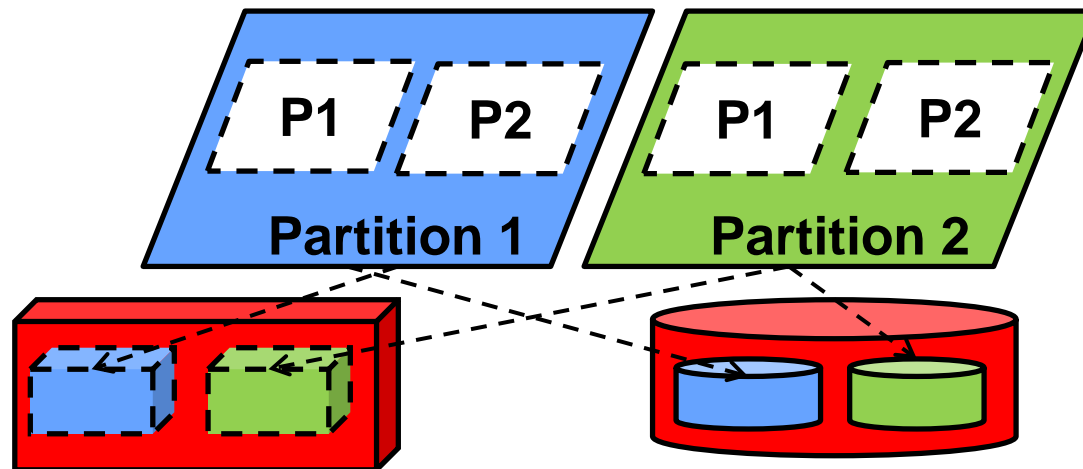
Use the regular process component

Include partition resources (tasks, data, etc.)

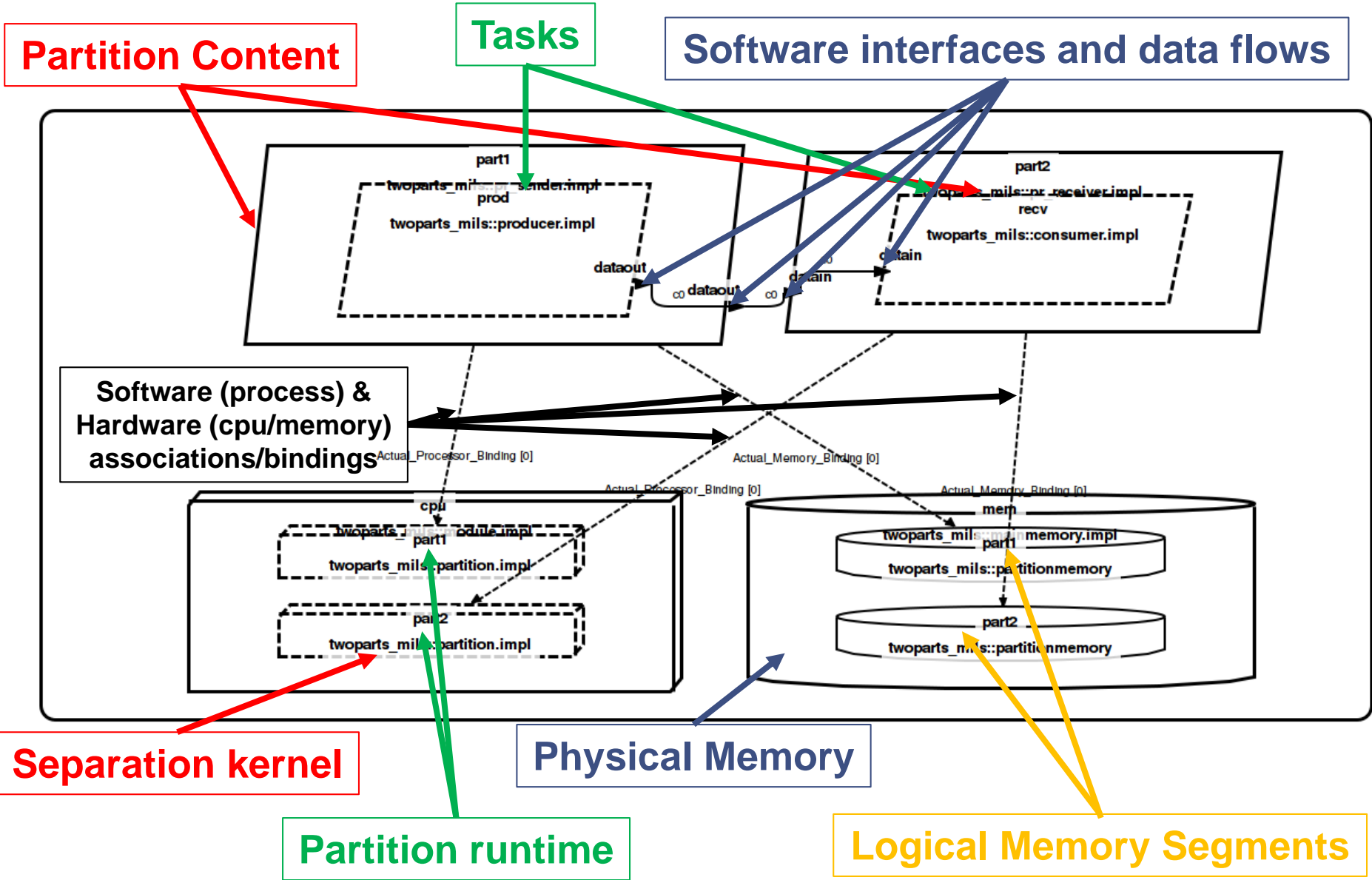
## Time and Space Isolation

Time: Partition execution slots

Space: Association of partitions to memory segments



# Modeling a MILS architecture - example



# Safety Policy with the Error-Model Annex V2

## Standardized AADL annex dedicated for safety specification

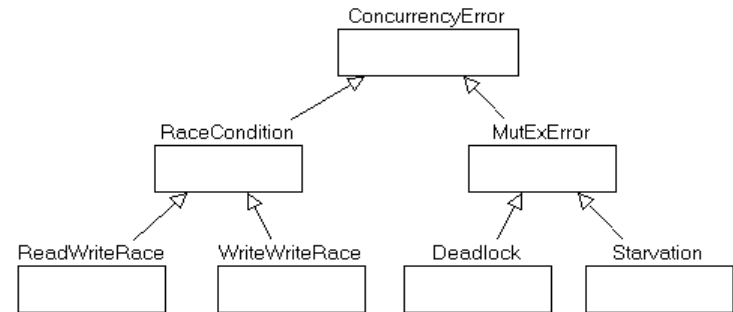
Integrated with AADL-core

Extend/refine existing models

## Support of Error Types Ontology

Characterize the error (i.e. divide by zero, late value)

Types hierarchy (i.e. late value is an extension of a timing error)



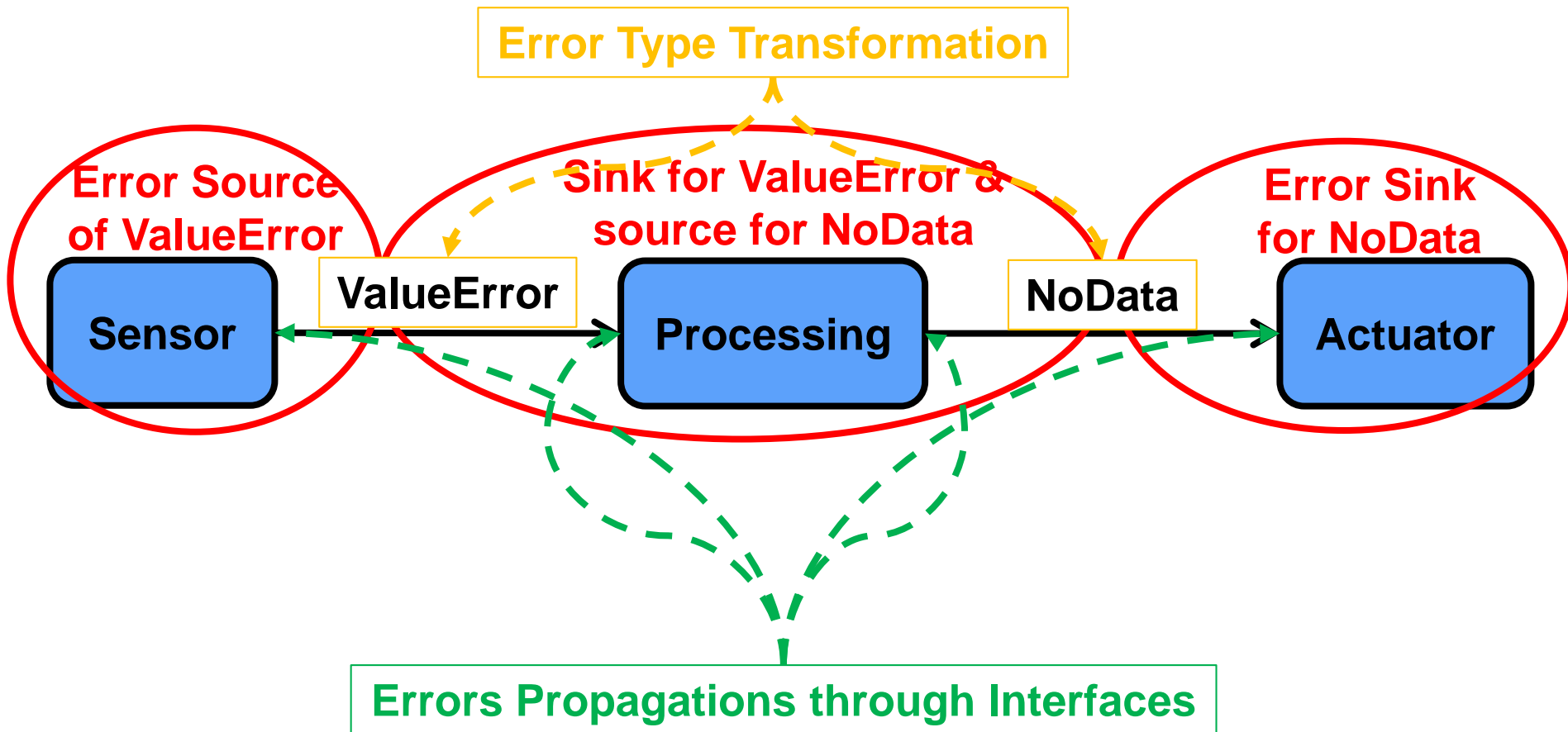
## Error Propagations and Behavior Specification

Errors being propagated by AADL components

Behavior based on external interfaces or sub-components

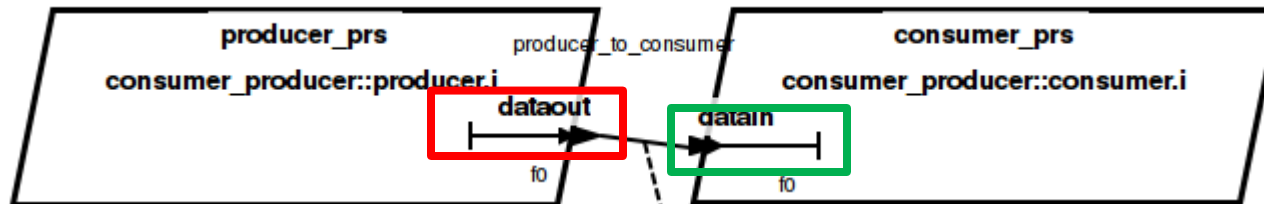


# Error Propagation





# Error Propagation Example



```
thread producer
features
```

```
  dataout : out data port Character;
```

```
annex EMV2 {**
```

```
  use types errorlibrary;
```

```
  use behavior errorlibrary::FailAndRecover;
```

```
  error propagations
```

```
    dataout : out propagation {ValueError};
```

```
  flows
```

```
    f0 : error source dataout {ValueError};
```

```
end propagations;
```

```
component error behavior
```

```
events
```

```
  ComputationError : error event;
```

```
transitions
```

```
  t0 : Operational -[ComputationError]-> Failed;
```

```
propagations
```

```
  p0 : Failed -[]-> dataout{ValueError};
```

```
end component;
```

```
properties
```

```
  EMV2::severity => ARP4761::Hazardous applies to dataout.ValueError;
```

```
  EMV2::OccurrenceDistribution => [ ProbabilityValue => 1.42e-5 ; Distribution => Poisson;]
```

```
  applies to dataout.ValueError;
```

```
  EMV2::likelihood => ARP4761::Probable applies to dataout.ValueError;
```

```
  EMV2::hazards =>
```

```
    ([ crossreference => "TBD";
```

```
      failure => "";
```

```
      phases => ("all");
```

```
      description => "Bad Value from the thread producer";
```

```
      comment => "Must check the software that the value is not faulty";
```

```
    ])
```

```
  applies to dataout.ValueError;
```

```
**};
```

```
thread consumer
```

```
features
```

```
  datain : in data port Character;
```

```
annex EMV2 {**
```

```
  use types errorlibrary;
```

```
  use behavior errorlibrary::FailAndRecover;
```

```
  error propagations
```

```
    datain : in propagation {ValueError};
```

```
  flows
```

```
    f0 : error sink datain {ValueError};
```

```
end propagations;
```

```
component error behavior
```

```
transitions
```

```
  t0 : Operational -[datain{ValueError}]-> Failed;
```

```
end component;
```

```
properties
```

```
  EMV2::severity => ARP4761::Hazardous applies to datain.ValueError;
```

```
**};
```

```
end consumer;
```



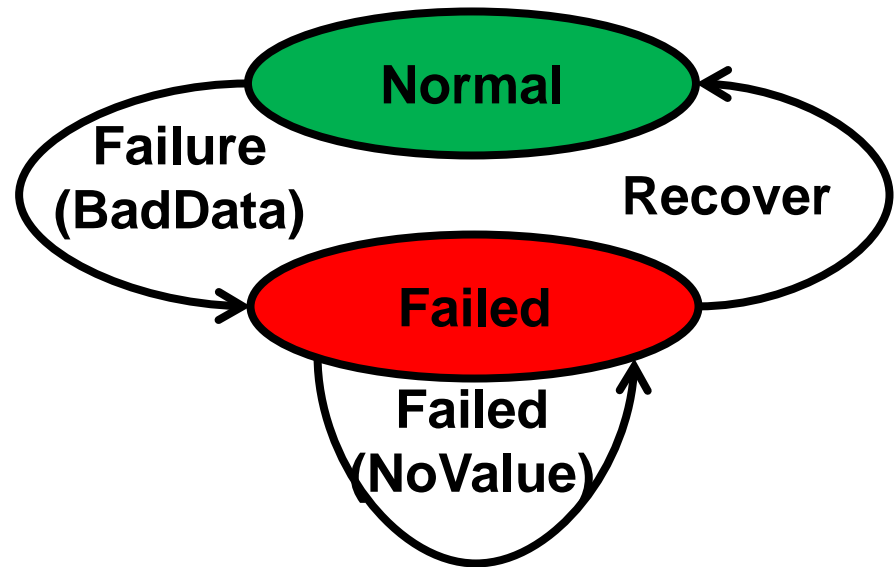
# Error behavior

States machines

Error-related transitions

Propagation rules

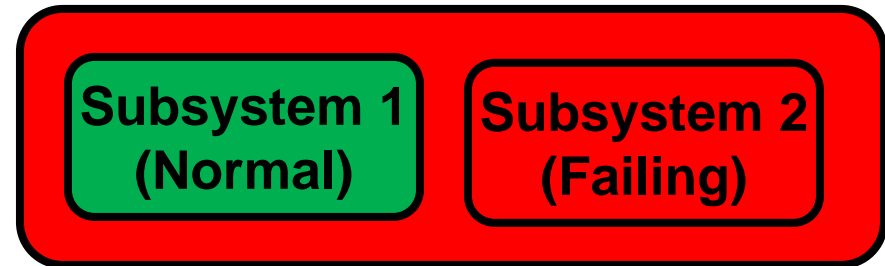
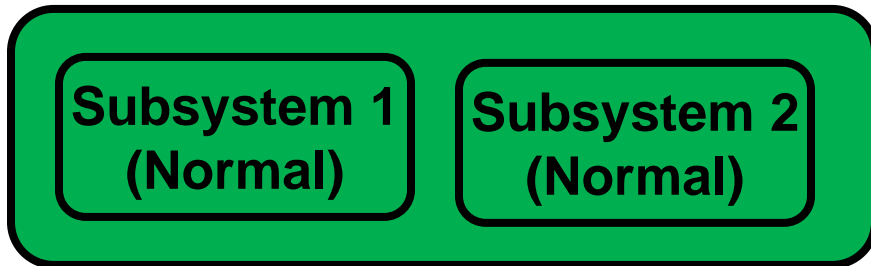
Use of error types



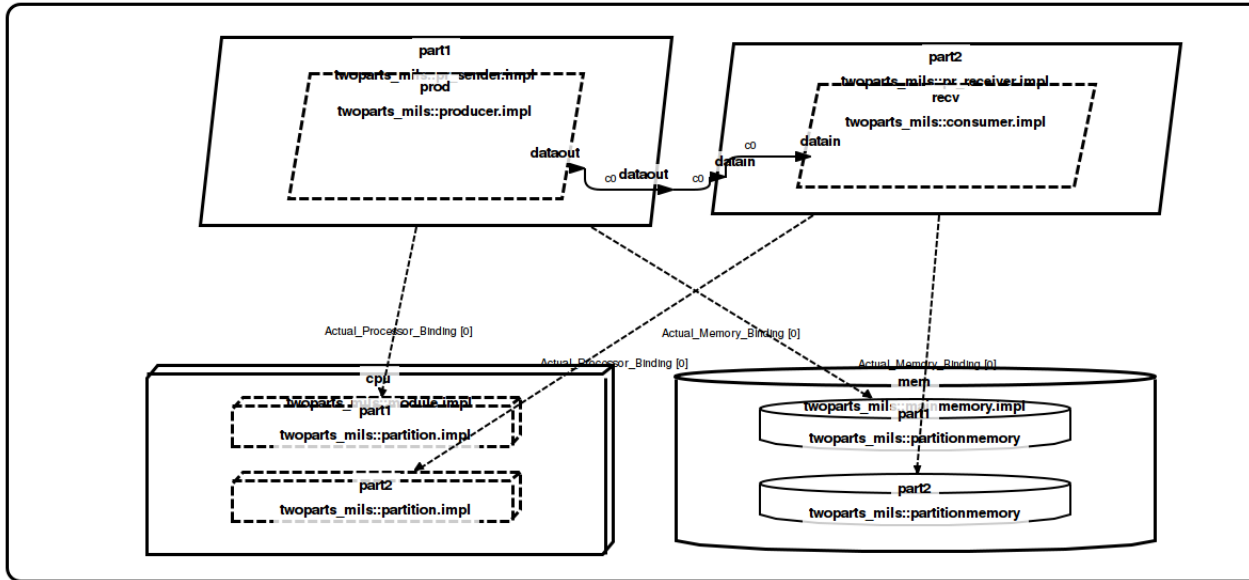
Composite behavior

Define system states according to its parts

*ex: "I am failing if one of my component is failing"*



# Error behavior example



```

cpu.part1)) applies to part1;
cpu.part2)) applies to part2;

t1;
t2;

```

```

annex EMV2 {
  -- prove (check_deos_compliance(this))
  prove (check_mils_compliance(this))
  **};
annex EMV2 {**
  use types errorlibrary;
  use behavior errorlibrary::FailAndRecover;

  composite error behavior
  states
    [part1.Failed]-> Failed;
    [part2.Failed]-> Failed;
    [cpu.Failed]-> Failed;
  end composite;
  **};
end node.impl;

```



# Agenda

Introduction to AADL

AADL modeling patterns for safety and security

**AADL validation tools dedicated to security and safety**

Demonstration



# Security Policy Verification

## Component integration and composition

Partitions share the same level with their tasks

Partitions contain objects at the same level

## Runtime issues

Each process is isolated in a partition

Partitions has at least one execution slot

Memory segments contain partitions at the same security level

## Communication Policies

Communication share the same level

A shared device manages objects at the same level



# Specifying Validation Rules with RESOLUTE

## Specify constraints on the AADL model

Check model consistency and properties

Validation at model level, avoid propagation of errors

## List of rules and functions to check the model

Select elements to be verified

Filter them according to your constraints

Check components characteristics

Select process, connections & virtual processor elements

```
check_mils_partitions_connections (s : system) <=
  ** "Check that connected partitions in " s " share the same security level" **
  forall (p1 : process) (p2 : process) (c : connection) (vp1 : virtual_processor) (vp2 : virtual_processor) .
  (connected (p1, c, p2)) and (processor_bound (p1, vp1)) and (processor_bound (p2, vp2))
  => property (vp1::SEI::SecurityLevel) = property (vp2, SEI::SecurityLevel)
```

Filter connected partitions  
with their associated runtime

Check the runtime security  
level is equal

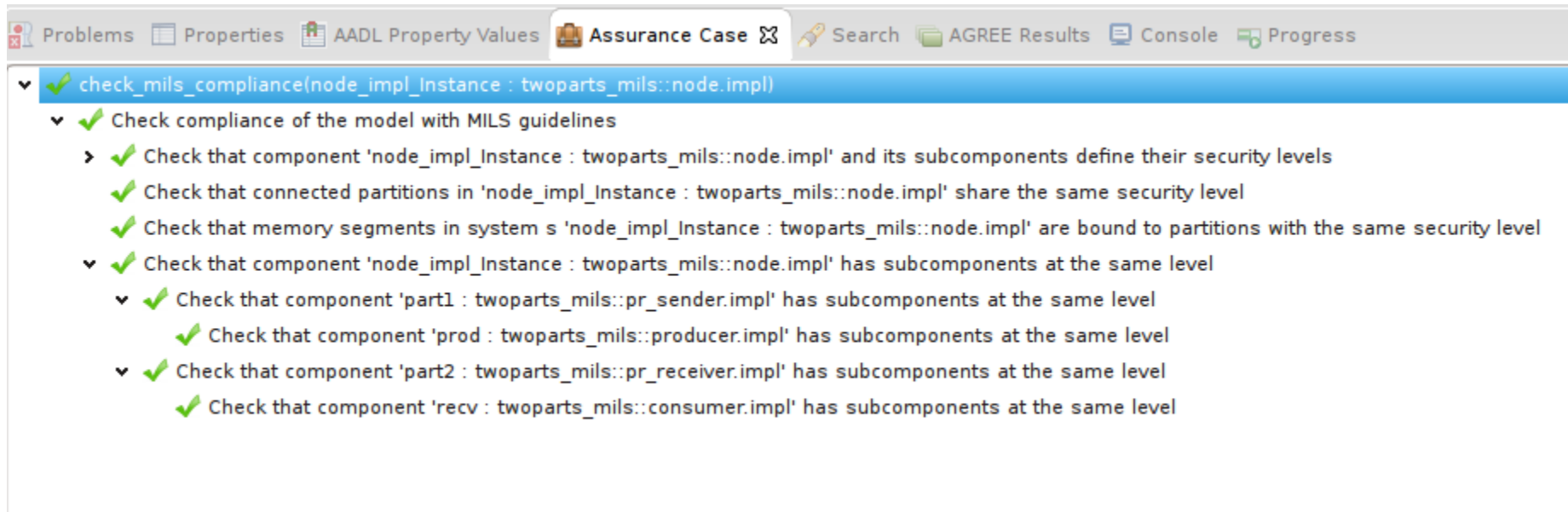


# Generating Assurance Cases

Generate assurance-cases using RESOLUTE and AADL

Show constraints dependencies

Export to Certware



The screenshot displays a software interface with a menu bar at the top containing: Problems, Properties, AADL Property Values, Assurance Case (selected), Search, AGREE Results, Console, and Progress. Below the menu bar, a tree view shows a list of assurance cases. The root node is 'check\_mils\_compliance(node\_impl\_Instance : twoparts\_mils::node.impl)', which is expanded to show a list of checks, all marked with a green checkmark. The checks are:

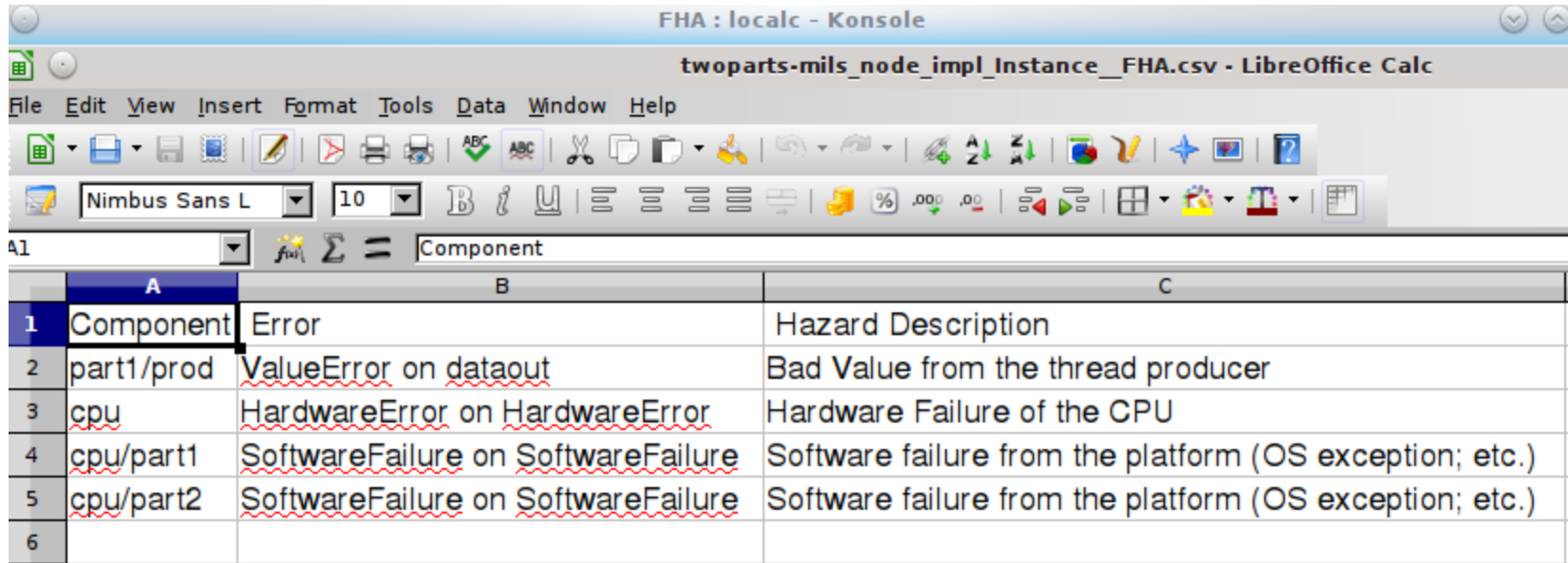
- Check compliance of the model with MILS guidelines
  - Check that component 'node\_impl\_Instance : twoparts\_mils::node.impl' and its subcomponents define their security levels
  - Check that connected partitions in 'node\_impl\_Instance : twoparts\_mils::node.impl' share the same security level
  - Check that memory segments in system s 'node\_impl\_Instance : twoparts\_mils::node.impl' are bound to partitions with the same security level
- Check that component 'node\_impl\_Instance : twoparts\_mils::node.impl' has subcomponents at the same level
  - Check that component 'part1 : twoparts\_mils::pr\_sender.impl' has subcomponents at the same level
    - Check that component 'prod : twoparts\_mils::producer.impl' has subcomponents at the same level
  - Check that component 'part2 : twoparts\_mils::pr\_receiver.impl' has subcomponents at the same level
    - Check that component 'recv : twoparts\_mils::consumer.impl' has subcomponents at the same level



# Safety documentation Generation - FHA

## Functional Hazard Assessment

List of all error sources of the system



	A	B	C
1	Component	Error	Hazard Description
2	part1/prod	<u>ValueError on dataout</u>	Bad Value from the thread producer
3	cpu	<u>HardwareError on HardwareError</u>	Hardware Failure of the CPU
4	cpu/part1	<u>SoftwareFailure on SoftwareFailure</u>	Software failure from the platform (OS exception; etc.)
5	cpu/part2	<u>SoftwareFailure on SoftwareFailure</u>	Software failure from the platform (OS exception; etc.)
6			



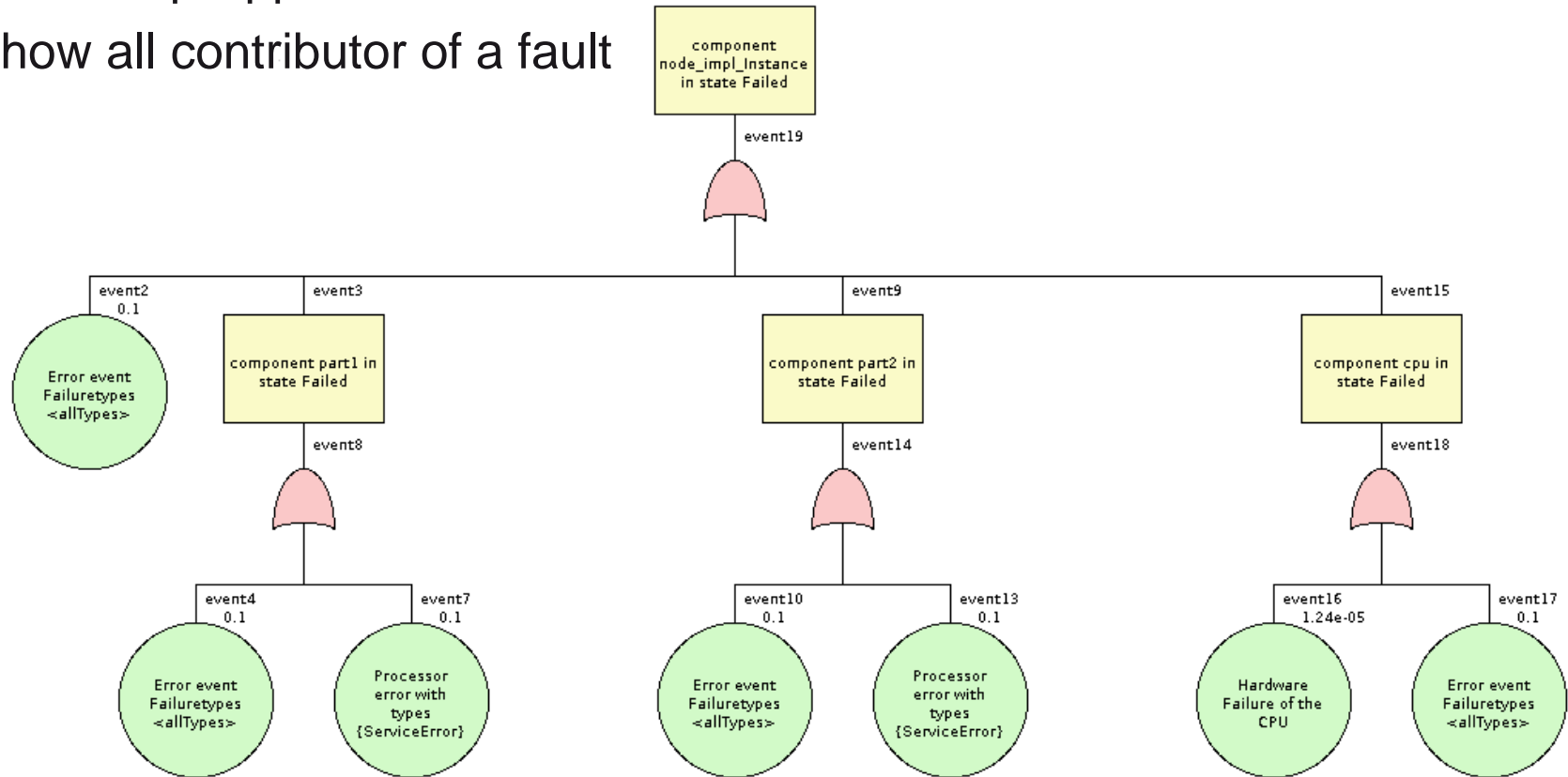


# Safety documentation Generation - FTA

## Fault-Tree Analysis

Bottom-up Approach

Show all contributor of a fault



# Safety documentation Generation – Fault Impact

## Failure Mode and Effect Analysis

Propagation paths of failures

Highlight failure containment

twoparts-mils\_node\_impl\_Instance\_FaultImpact.csv - LibreOffice Calc

	A	B	C	D	E	F
1						
2						
3	<b>Component</b>	<b>Initial Failure Mode</b>	<b>1st Level Effect</b>	<b>Failure Mode</b>	<b>second Level Effect</b>	<b>Failure Mode</b>
4	part1.prod	{ValueError}	{ValueError} dataout -> part2.recv:datain	part2.recv {ValueError} [Masked]		
5	cpu.part1	{ServiceError}	{ServiceError} bindings -> part1:processor	part1 {ServiceError}	{ValueError} dataout ->	part2.recv {ValueError} [Masked]
6	cpu.part2	internal event Fail	{ItemOmission} bindings -> part2:processor	part2 {ItemOmission} [Failure Effect]		
7	cpu.part2	internal event Soft	{LateServiceTermination} bindings -> part2:processor	part2 {LateServiceTermination} [Failure Effect]		
8	cpu.part2	{ServiceError}	{ServiceError} bindings -> part2:processor	part2 {ServiceError} [Failure Effect]		
9						



# Automatic Code Generation

## **Automatically produce system implementation**

Ensure implementation of system requirements

Avoid traditional mistakes of manual code generation

## **Low overhead (memory footprint and additional CPU time)**

Less than 10% in memory and computation increase

Benefits outweigh the potential

## **Support for different runtime**

ARINC653/MILS – focus on safety/security (DeOS, POK)

POSIX (RTEMS, Linux)



# Agenda

Introduction to AADL

AADL modeling patterns for safety and security

AADL validation tools dedicated to security and safety

Demonstration



# Conclusion

## **AADL flexible language to define safety and security concerns**

Early verification, reducing tests and integration costs

Automatic code production, avoiding code and integration mistakes

## **Integration with existing development methods**

Safety documentation (i.e. ARP4761)

Coding standards (i.e. ARINC653)

## **Bridge with Validation and Assurance Case tools**

Check model consistency and composition

Auto-Generate assurance cases from models



# Links & Useful Information

AADL website – <http://www.aadl.info>

AADL wiki – <http://www.aadl.info/wiki>

ARINC653 AADL annex standard - <http://standards.sae.org/as5506/2/>



# Contact

## Dr. Julien Delange

RTSS AP Initiative

Telephone: +1 412-268-9652

Email: [jdelange@sei.cmu.edu](mailto:jdelange@sei.cmu.edu)

## Web

[www.aadl.info](http://www.aadl.info)

[www.sei.cmu.edu](http://www.sei.cmu.edu)

[www.sei.cmu.edu/contact.cfm](http://www.sei.cmu.edu/contact.cfm)

## U.S. Mail

Software Engineering Institute

Customer Relations

4500 Fifth Avenue

Pittsburgh, PA 15213-2612

USA

## Customer Relations

Email: [info@sei.cmu.edu](mailto:info@sei.cmu.edu)

Telephone: +1 412-268-5800

SEI Phone: +1 412-268-5800

SEI Fax: +1 412-268-6257

