

# Architectural Security Modeling with the AADL

**Jorgen Hansson & Peter Feiler**  
Email: {hansson,phf}@sei.cmu.edu  
Software Engineering Institute  
Carnegie Mellon University, USA

**Aaron Greenhouse**  
SureLogic Inc, USA



# SEI Security Framework in AADL

---

## Security framework features:

- representation of confidentiality requirements of resources (i.e., objects)
- representation and generation of security clearance/privileges of subjects operating on the objects
- representation of access matrix, specifying allowed access operations of subjects on objects to support integrity
- analysis of an Architecture Analysis and Design Language (AADL) model system with respect to basic confidentiality principle, need-to-know principle, least privileges, and controlled sanitization.
- supports MLS and Bell-LaPadula based frameworks



# Outline

---

- Validating security/confidentiality of a software architecture
  - Representation of a well-known security framework Bell-LaPadula in AADL
  - Analysis in OSATE
- MILS Systems
- Conclusions



# What can MBE and AADL bring to the security arena?

- Validating security, e.g., confidentiality, integrity
  - Security requirements and clearances
- Validating the security architecture

Application &  
user perspective

## Validating the mapping of applications to systems

- Validating effects of security on other system behavior
  - Avoiding undesirable side-effects
  - Trade-off analysis among quality attributes

perspective



# MBE Security Approach using AADL

---

## Representation of

- confidentiality requirements of resources (i.e., objects)
- security clearance/privileges of subjects operating on the objects
- access matrix, specifying allowed access operations of subjects on objects to support integrity

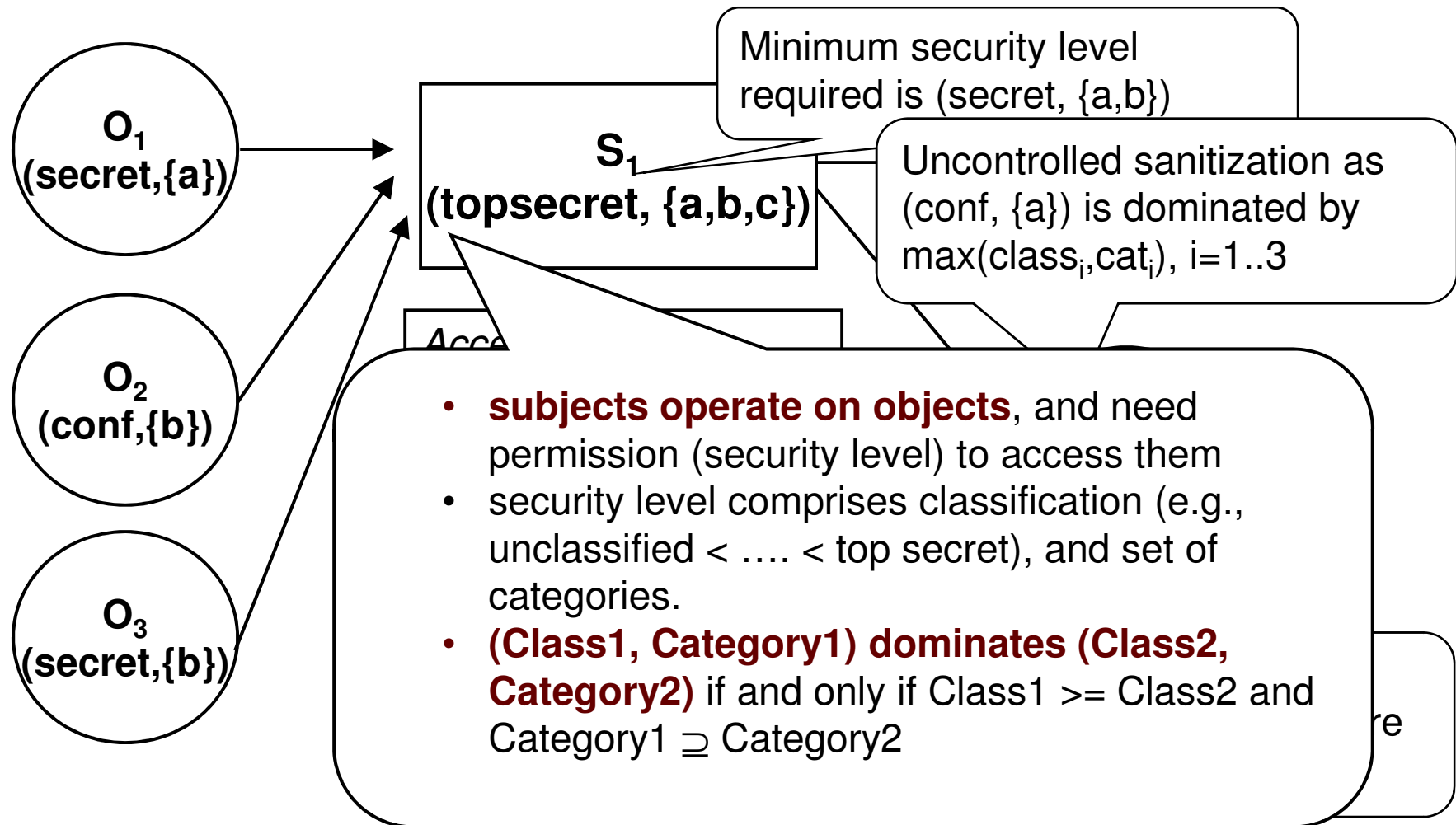
Analysis of confidentiality, star property, need-to-know principle, least privileges, and controlled sanitization

- Distinguish errors from warnings

Supports MLS and Bell-LaPadula based frameworks



# Confidentiality Preliminaries



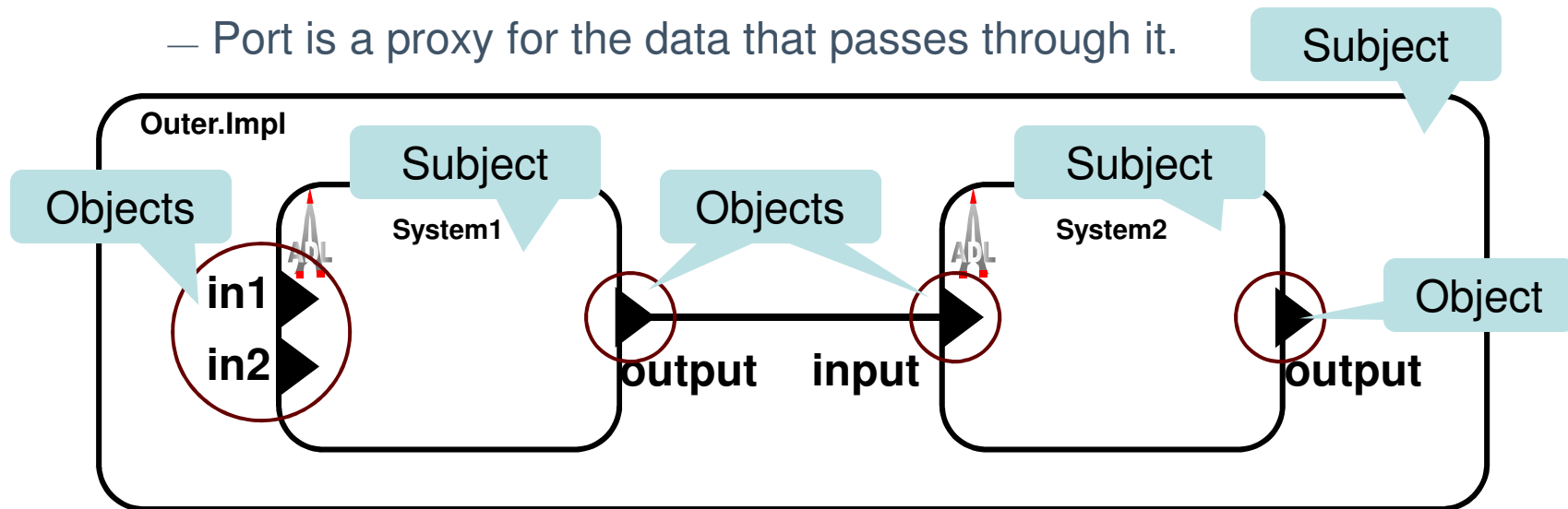
# Subjects and Objects in AADL

AADL components pass data through ports and over connections.

- Data is not represented explicitly.

Mapping Bell–La Padula to AADL:

- Subject → Component
- Object → Port Feature
  - Port is a proxy for the data that passes through it.



# Definition of the Security Types

---

```
-- Property intended to be customized by modelers.  
-- Parameterizes the security property definitions.  
property set Security_Types is  
  -- Military levels by default
```

```
  Classifications:
```

```
    type enumeration (unclassified, confidential, secret,  
                     top_secret);
```

```
-- This must be the first element of Classifications
```

```
  Default_Classification:
```

```
    constant Security_Types::Classifications =>  
      unclassified;
```

```
-- Default set of categories
```

```
  Categories:
```

```
    type enumeration (A, B, C, D);
```

```
end Security_Types;
```



# Definition of the Security Properties 1

---

```
property set Security_Attributes is
  Class: inherit Security_Types::Classifications =>
    value(Security_Types::Default_Classification)
    applies to (data, subprogram, thread, thread group,
                process, memory, processor, bus, device,
                system, port, server subprogram,
                parameter, port group);

  Category: inherit list of Security_Types::Categories =>
    ()
    applies to (data, subprogram, thread, thread group,
                process, memory, processor, bus, device,
                system, port, server subprogram,
                parameter, port group);

  -- ...
end Security_Attributes;
```



# Definition of the Security Properties 2

---

```
property set Security_Attributes is
```

```
  Class: inherit Security_Types::Classifications =>  
    value(Security_Types::Default_Classification)
```

```
    applies to (data, subprogram, thread, thread group,  
               process, memory, processor, bus, device,  
               system, port, server subprogram,  
               parameter, port group);
```

```
  Category: inherit list of Security_Types::Categories =>  
    ()
```

```
    applies to (data, subprogram, thread, thread group,  
               process, memory, processor, bus, device,  
               system, port, server subprogram,  
               parameter, port group);
```

```
-- ...
```

```
end Security_Attributes;
```

Properties apply to all component and feature categories;  
i.e., all subjects and objects.



# Definition of the Security Properties 3

---

```
property set Security_Attributes is
  Class: inherit Security_Types::Classifications =>
    value(Security_Types::Default_Classification)
    applies to (data, subprogram, thread, thread group,
                process, memory, processor, bus, device,
                system, port, server subprogram,
                parameter, port group);

  Category: inherit list of Security_Types::Categories =>
    ()
    applies to (data, subprogram, thread, thread group,
                process, memory, processor, bus, device,
                system, port, server subprogram,
                parameter, port group);

  -- ...
end Security_Attributes;
```

**Usability:** Default security level is the lowest level.



# Definition of the Security Properties 4

---

```
property set Security_Attributes is
  Class: inherit Security_Types::Classifications =>
    value(Security_Types::Default_Classification)
    applies to (data, subprogram, thread, thread group,
                process, memory, processor, bus, device,
                system, port, server subprogram,
                parameter, port group);

  Category: inherit list of Security_Types::Categories =>
    ()
    applies to (data, subprogram, thread, thread group,
                process, memory, processor, bus, device,
                system, port, server subprogram,
                parameter, port group);

  -- ...
end Security_Attributes;
```

**Usability:** If not otherwise specified, subcomponents and features inherit their security level from their container.  
**Creates safe default property values with minimal work.**



# Modeling and Validation: Software Level Scenarios

---

1. Derive minimum security clearance of processes/threads based on confidentiality requirements of ports
2. Enforce that confidentiality requirements of data elements (ports) are satisfied by the security clearance of processes/threads
3. Derive access control matrix based on the security levels of processes/threads and data elements
4. Strong validation by detailed analysis of security levels of processes/threads and data elements and the access control rules represented as a matrix



# Modeling and Validation: Hardware/Software Level

---

Determine the viability of a system (software architecture mapped to hardware) given confidentiality requirements of data objects and security clearance by users

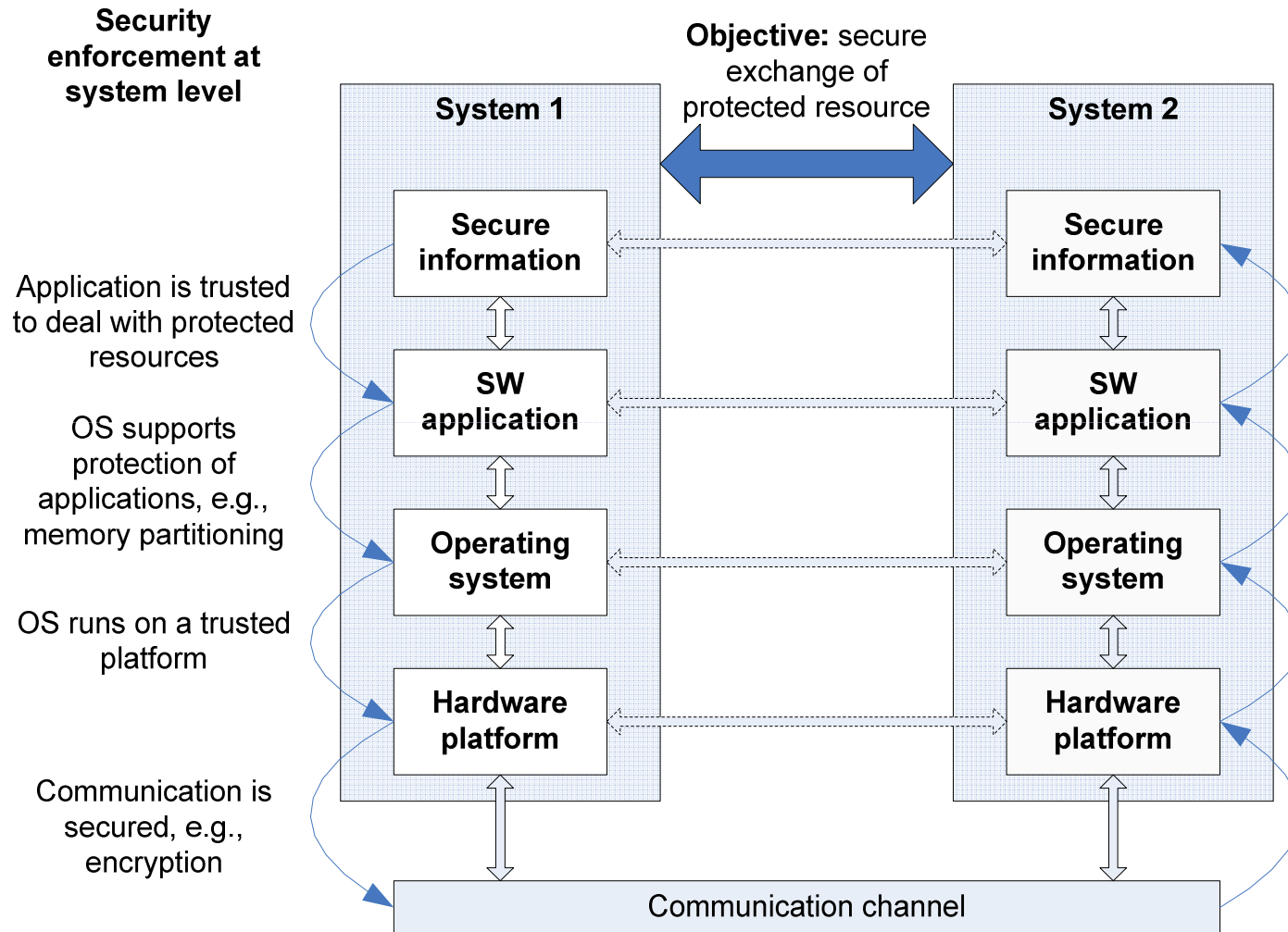
- Processors, memory, bus, processes, threads

Analysis:

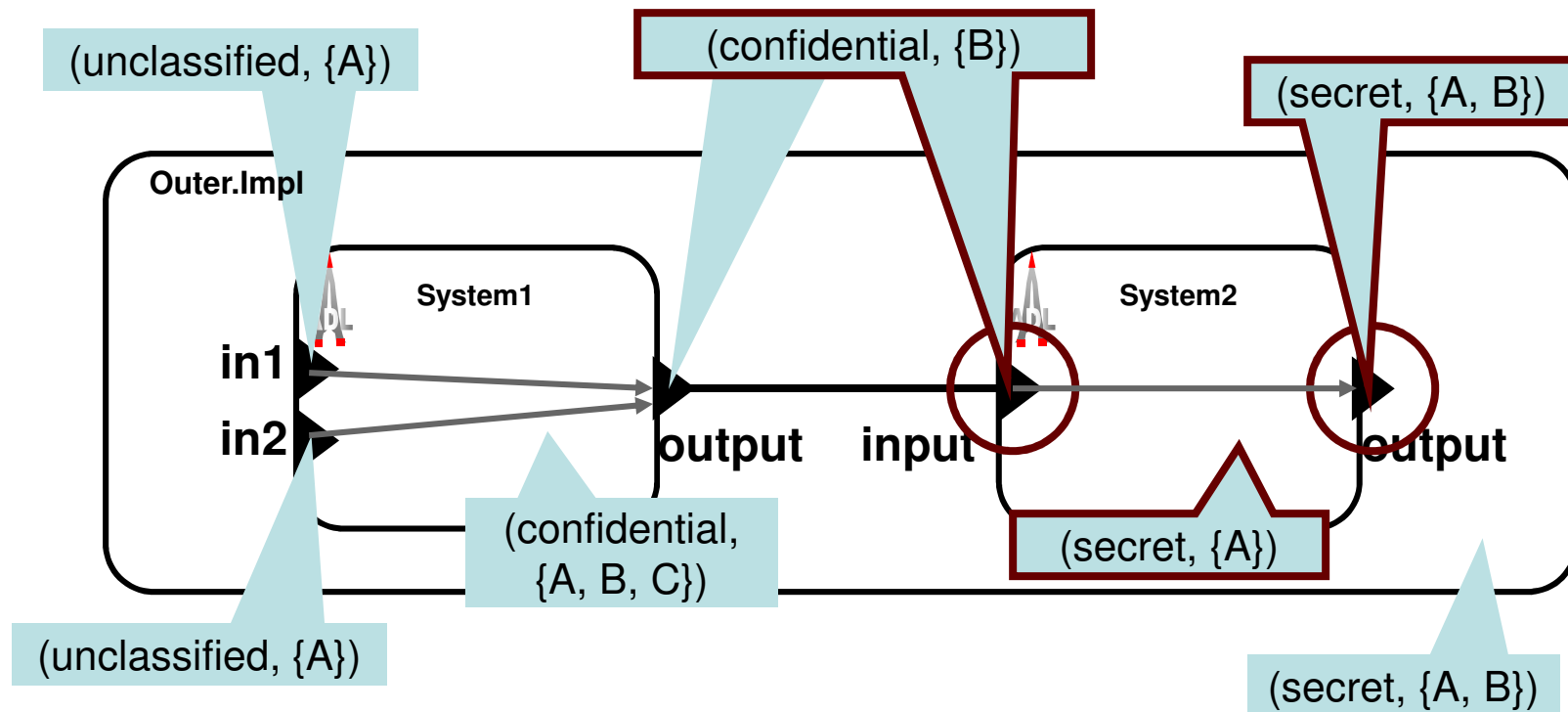
- Ensure processes and threads are mapped to appropriate hardware, communicate over secured channels, and reside/store data in protected memory
- Derive minimum security requirements on hardware components given a software architecture



# System-level security



Problems		AAADL Property Values	Properties	Error Log	Progress
5 errors, 1 warning, 0 infos					
Description	Resource				
<b>Architectural Errors (1 of 1 items)</b>					
✘ The security level of subcomponent "sub1" in Outer.Impl, (confidential, {A, B, C}), is not dominated by the security level, (secret, {A, B, C}), of its containing component. Example-with-flows.aad					
<b>Simple Security Property Violations (2 of 2 items)</b>					
✘ The security level of port "output" of System2, (secret, {A, B}), is not dominated by the level, (secret, {A}), of its containing component. Example-with-flows.aad					
✘ The security level of port "input" of System2, (confidential, {B}), is not dominated by the level, (secret, {A}), of its containing component. Example-with-flows.aad					
<b>Star Property Violations (2 of 2 items)</b>					
✘ The security level of source feature "in2" of flow path "f2" in System1, (unclassified, {A}), is not dominated by the security level, (secret, {A, B, C}), of its target feature. Example-with-flows.aad					
✘ The security level of source feature "in1" of flow path "f1" in System1, (unclassified, {A}), is not dominated by the security level, (secret, {A, B, C}), of its target feature. Example-with-flows.aad					
<b>Least Privilege Violations (1 of 1 items)</b>					
⚠ The security level of component type System1 is (confidential, {A, B, C}) but only needs to be (confidential, {A, B}). Example-with-flows.aad					



# Sanitization (lowering of security levels)

---

Enforcing a security model prevents information from being released to those not trusted to see it.

- Bell–La Padula star property: data can only become more secure.

This is too limiting; need to be able to **write down** or **sanitize** data:

- Need to be able to derive less secret data from secret data.
  - E.g., by obfuscating identifying information from a record
- Need to be able to pass secrets over public channels.
  - E.g., by encrypting the data first
- These valid exceptions violate the star property.

Models need to accommodate sanitization as an exception.

- Should document **where/when** sanitization is intended.



# MILS Architectures Rationale

---

- Reduction in physical hardware.
- Easier control and management of information among various communities of interest.
- Cheaper development of highly secure systems, as well as a faster time-to-market.
- Overall increase in safety.
- Less need for re-architecting systems to meet security standards.

*“Dramatically reduce the amount of safety/security critical code, so that one can dramatically increase the scrutiny of critical code. Reducing the size allows formal analysis to be conducted that otherwise would be deemed intractable”*



# MILS partners

---

- United States Air Force
  - AFRL (lead)
- United States Army
- United States Navy
- National Security Agency
- Boeing
- Lockheed Martin
- Objective Interface Systems
- Green Hills Software
- Lynux Works
- Wind River General Dynamics,
- Raytheon
- Rockwell Collins
- MITRE
- University of Idaho



# MILS Architectural Components

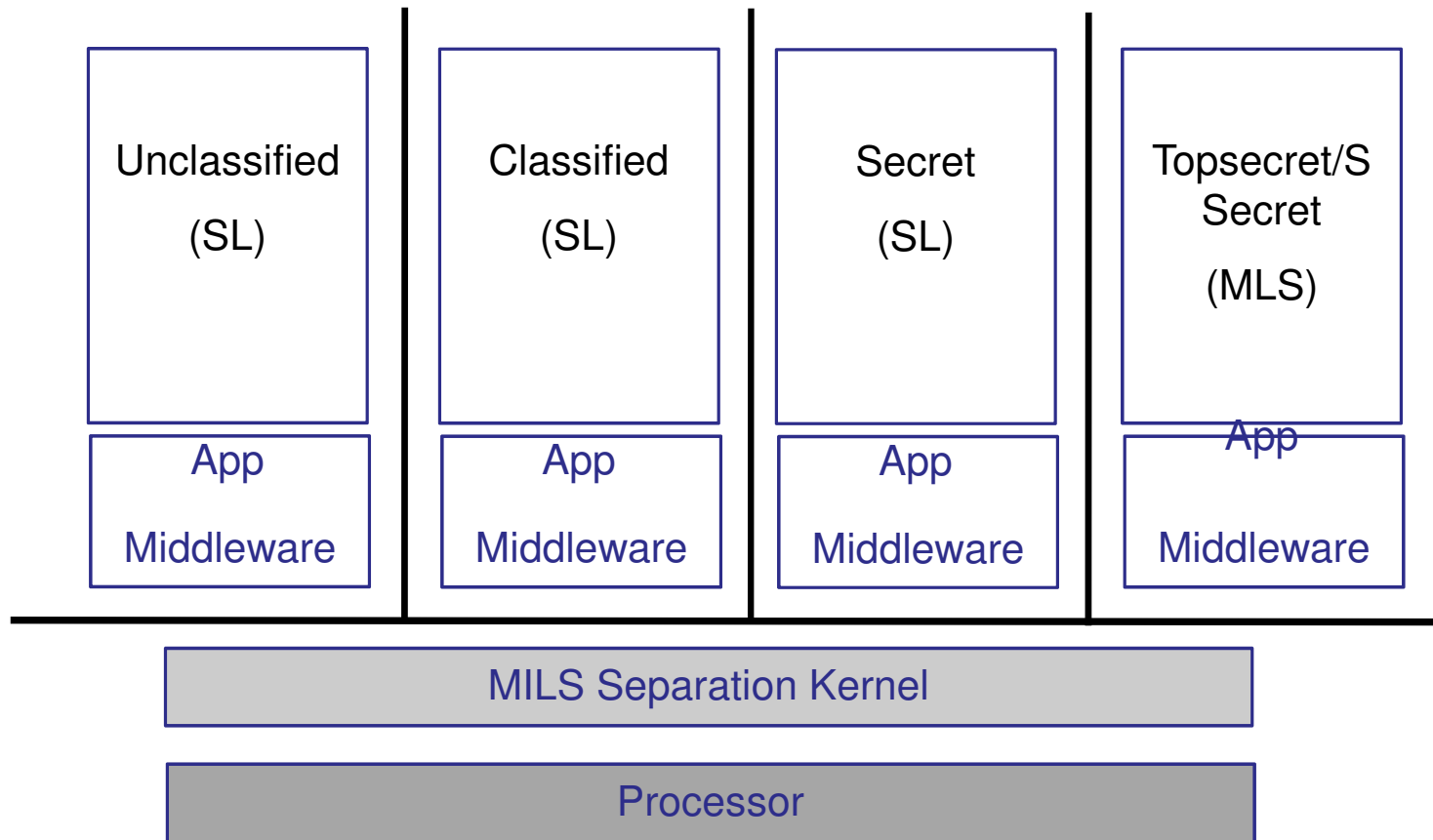
---

MILS architecture utilizes partitions to isolate processes. Each partition

- defines a collection of data objects, code, and system resources (partition can be evaluated separately)
- is divided into the following layers (each layer is responsible for its own security domain and nothing else):
  - Separation Kernel
  - Middleware Service layer
  - Application layer



# MILS Architecture



# MILS NEAT Requirements

---

- **Non-bypassable** -- security functions cannot be circumvented.
- **Evaluatable** -- size and complexity of the security functions allow them to be verified and evaluated.
- **Always invoked** -- security functions are invoked each and every time without exceptions.
- **Tamperproof** -- subversive code cannot alter the function of the security functions.



# Validation of MILS Architecture (1)

---

- A MILS architecture supports MLS
  - use of partitioning and separation
  - reduced complexity of the system is conducive to improved efficiency of certification
- Confidence in security validation increases with the level of decomposition, given that
  - (i) refinement patterns ensure that security is enforced, recognizing intra- and inter-level security requirements, and



# Validation of MILS Architecture (2)

---

Modeling and validation of security/confidentiality attributes, which includes MLS and Bell-LaPadula properties and additional specified security design principles

Architectural modeling and validation of assumptions underlying MILS:

- Assumption: **Damage limitation** and **partitioning** in MILS
- Assumption: Validation of **separation in time**

Mapping of software to hardware (behavior anomalies)

Enforcement of NEAT requirements (specifically N & A)

**Impact analysis:** evaluation of a MILS architecture configuration with respect to impact on other non-functional attributes



# Architecture Refinement Patterns

---

## SLS -Single-Level Secure Component

- has intra-level requirements
- only processes data at **one security level**

## MSLS –Multiple Single-Level Secure Component

- has intra-level requirements for each security level
- has MSLS inter-level requirement that there is **no** cross-level information flow

## MLS -Multi-Level Secure Component

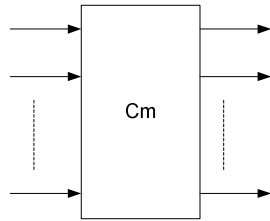
- has intra-level requirements for each security level
- has MLS inter-level requirements to **regulate** cross-level information flow



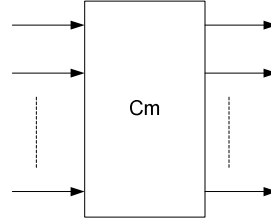
# Architecture Refinement Patterns

A component can be decomposed through product, cascade, or feedback

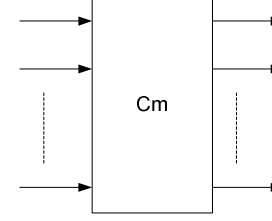
Each component can be SLS, MLS, or MSLS



pre



pre

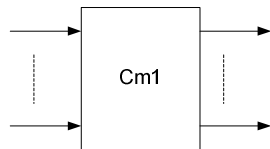
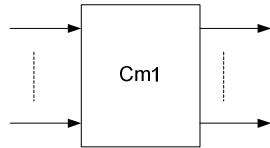


pre

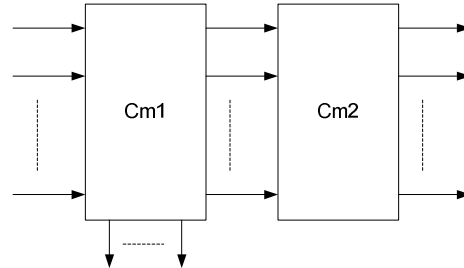
post

post

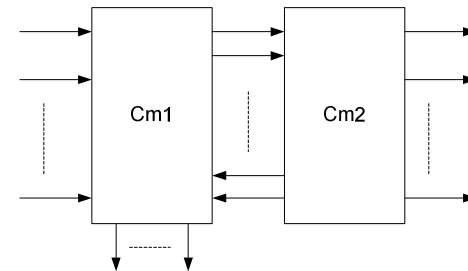
post



Product



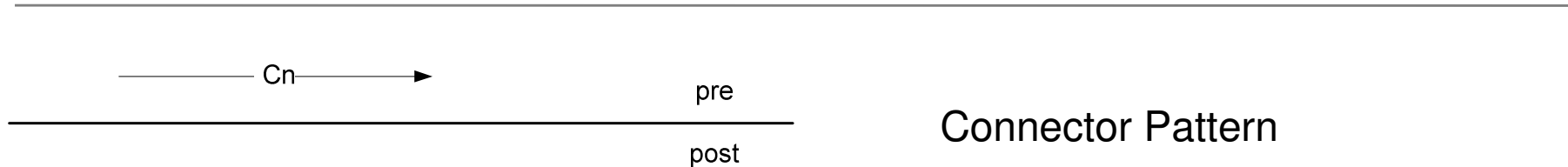
Cascade



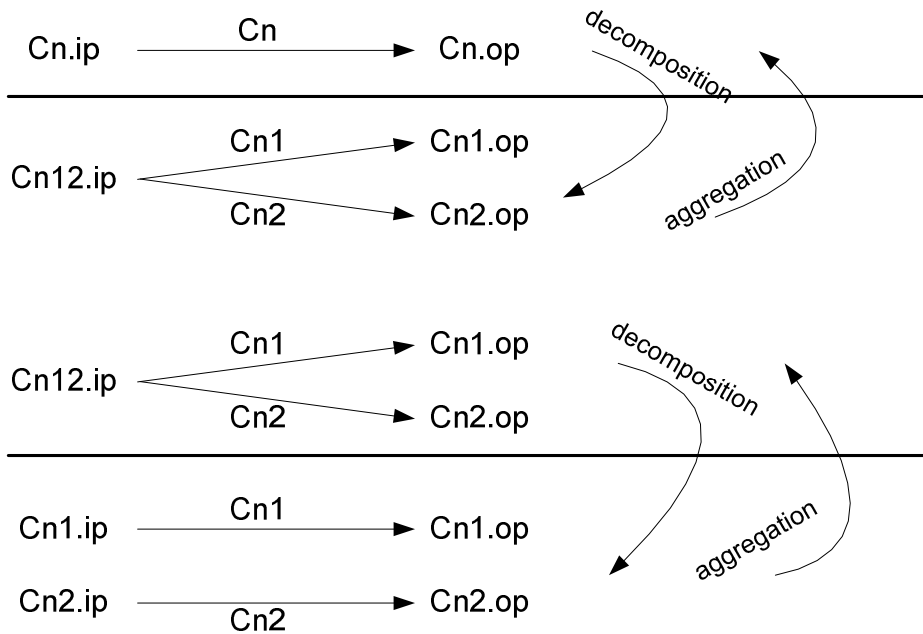
Feedback



# Port Decomposition/Aggregation Patterns



Connector Pattern



Port Patterns

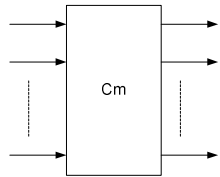
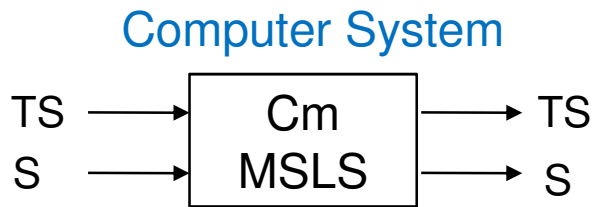


# Decomposition Patterns

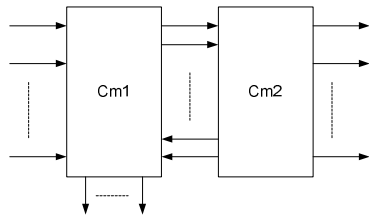
Component decomposition patterns for designing MLS systems							
Decomposition patterns	Cm	Policy types of components					
		Cm1			Cm2		
		SLS	MSLS	MLS	SLS	MSLS	MLS
Product	SLS	x			x		
	MSLS	x			x		
	MLS		x			x	
Cascade	SLS	x		x	x		x
	MSLS		x	x*		x	x*
	MLS	x		x*			x
Feedback	SLS	x		x	x		x
	MSLS		x	x*		x	x*
	MLS	x		x*	x		x



# A Decomposition Example



pre  
post

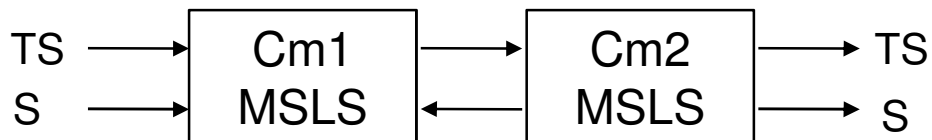


Rule:

MSLS =

+

MSLS + MSLS

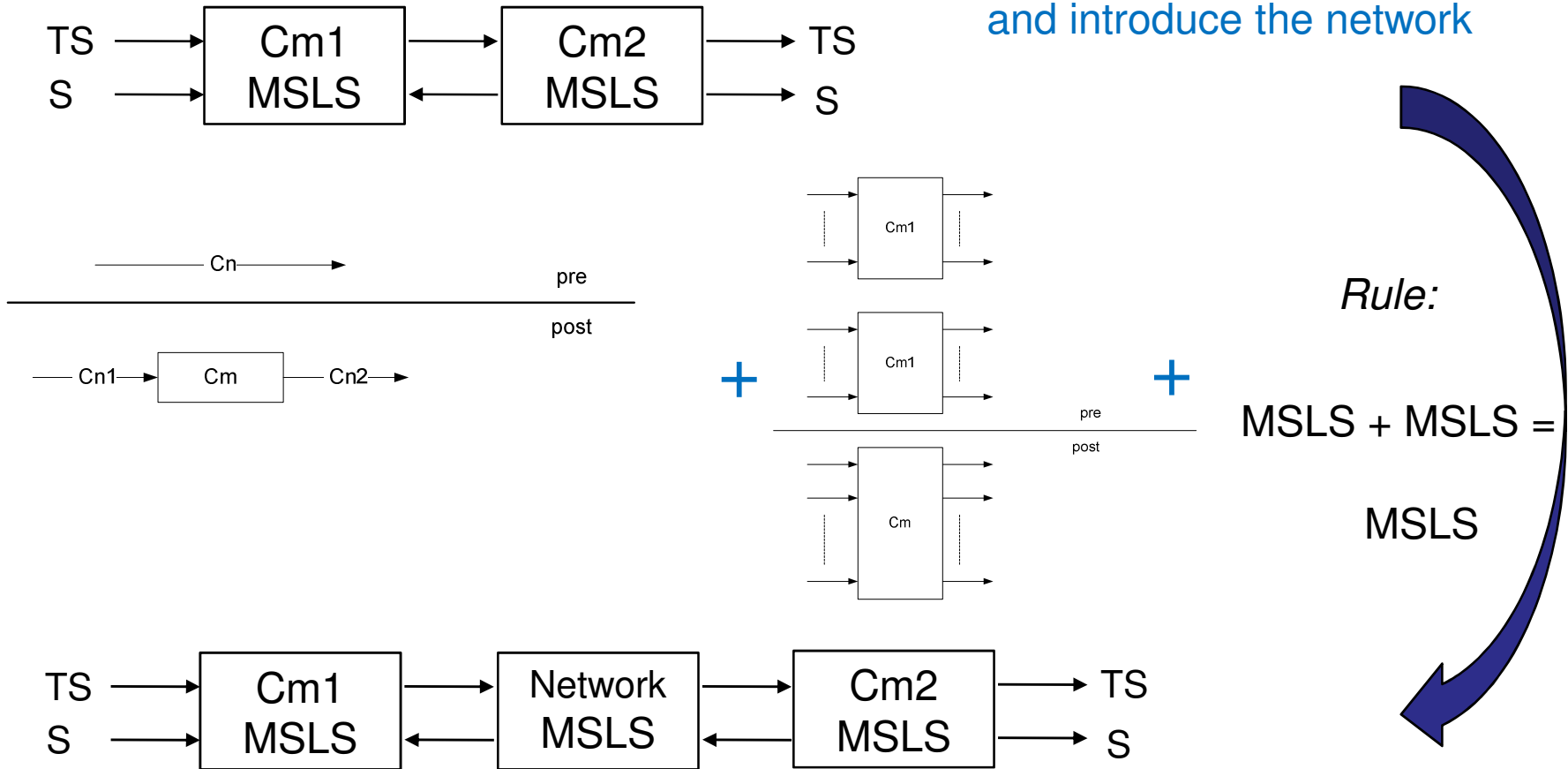


System consists of two interconnected sub-systems with bidirectional communication



# A Decomposition Example

Let us decompose further and introduce the network

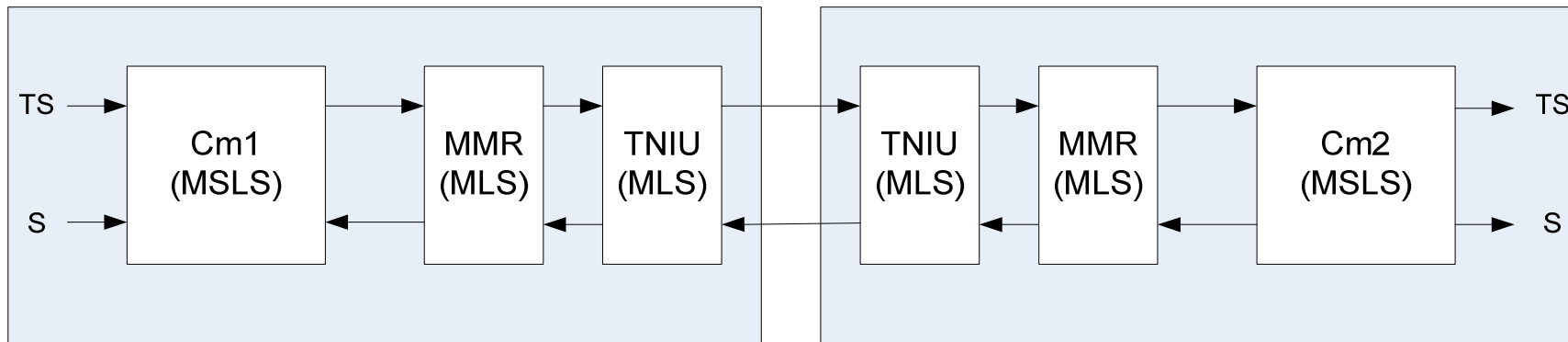


# A Decomposition Example

Further decomposition of the network, incorporating

- MILS Message Routing and (MMR)
- Trusted Network Interface Unit (TNIU)

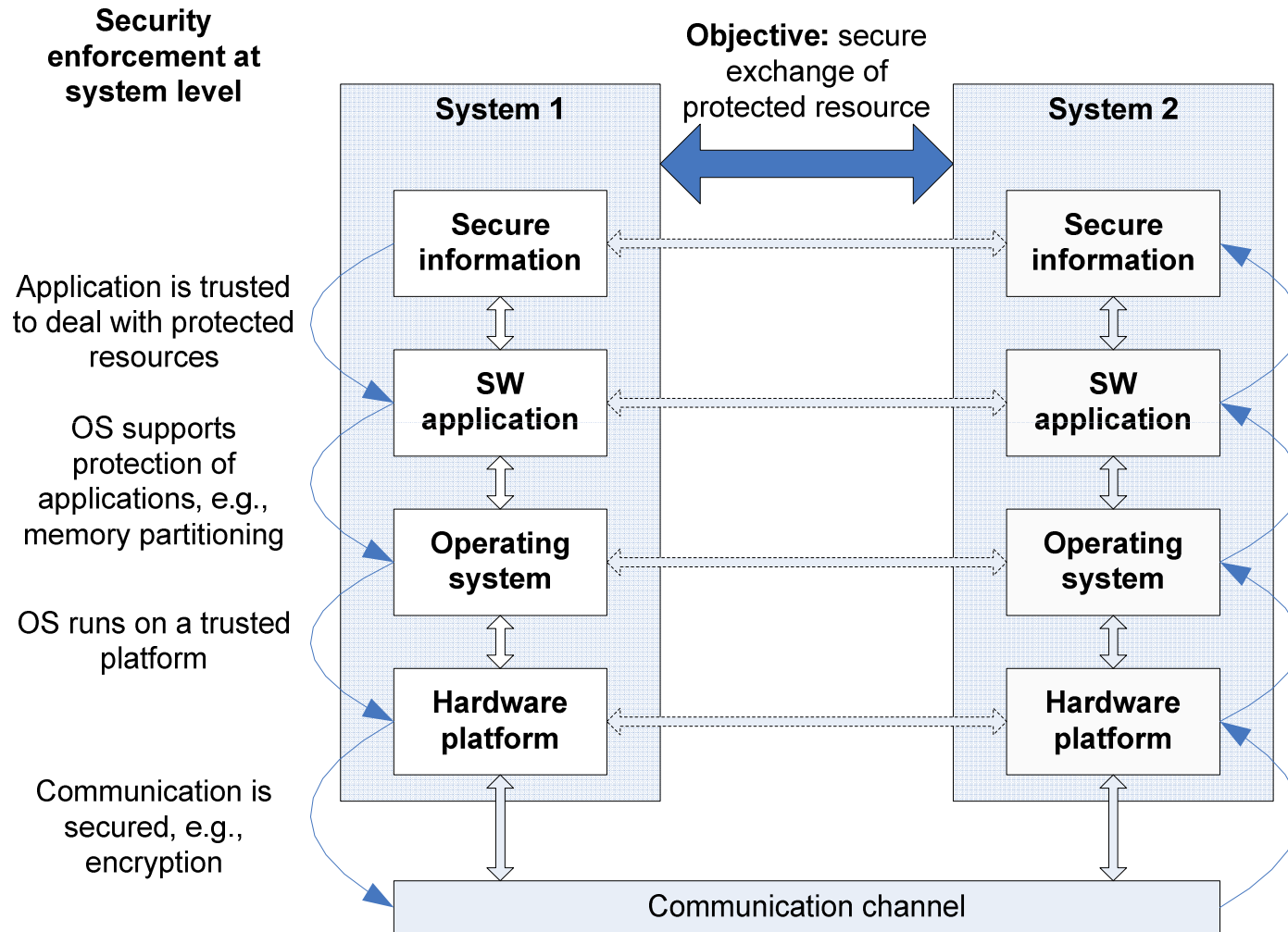
gives:



Next step: Decompose system Cm1 and Cm2 into Sw (applications, threads, partitions etc.) and hardware



# Which brings us back to...



# Summary

---

We have developed a framework to model and validate an architecture with respect to data quality attributes in AADL

- Determine an architecture is **secure**, i.e., it does not compromise confidentiality and integrity, and that sanitization can be performed correctly.
- Determine that data used by applications is **not compromised (security)**, **temporally valid**, and of adequate **precision** and **confidence**.
- Validate that applications under different modes (scenarios) use data of sufficient quality, e.g., compare normal operating mode, failure mode, and overload mode scenarios.



---

Thank you for your attention!



**Software Engineering Institute**

**CarnegieMellon**

For more information contact [hansson@sei.cmu.edu](mailto:hansson@sei.cmu.edu)  
or see: [www.sei.cmu.edu](http://www.sei.cmu.edu) and [www.aadl.info](http://www.aadl.info)



**Software Engineering Institute**

**CarnegieMellon**

© 2009 Carnegie Mellon University