Actionable Definition of Safety Design Patterns Using AADLv2, ALISA, and the Error Modeling Annex

Jerome Hugues

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

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.

DM20-0759

Actionable safety patterns – ISSE project

ISSE project: interplay between safety and security.

- Hazard analysis
- Extended analysis for safety and security, with Kansas State U. (J. Hatcliff group)
- Defining safety and security policies as patterns

Literature on safety patterns keeps being informal, e.g. [1]

- Negative impact on reuse of established expertise and practice

Today: Definition of safety patterns as library of models & verification plans using AADL & additional notations: ALISA, AWAS & AGREE

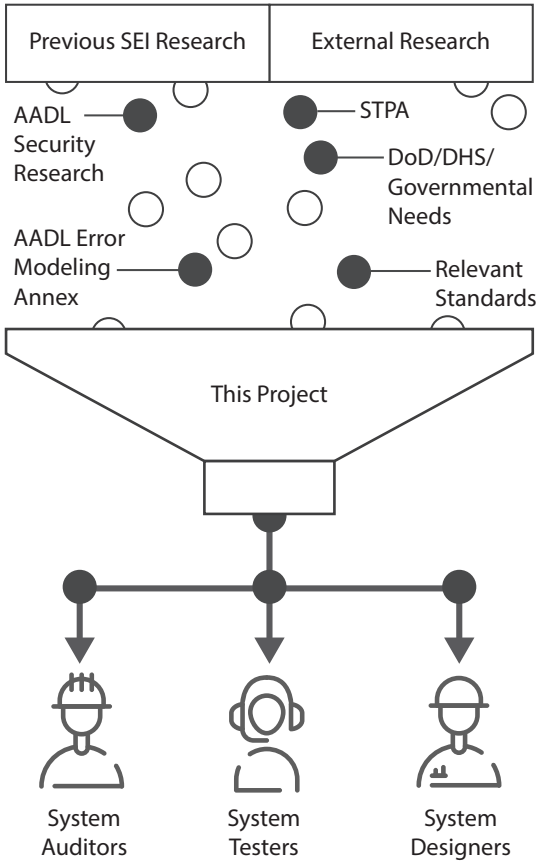
[1] C. Preschern et al. *"Safety Architecture Pattern System with Security Aspects"*, https://doi.org/10.1007/978-3-030-14291-9_2

Making Critical Systems Safer and More Secure

Modern embedded systems need to be both safe and secure. As we have seen, the pace and scale of the development of these systems means traditional methods cannot keep up.

Research to Practice

The SEI works to rapidly move ideas from research in embedded systems – conducted either here at the SEI, in academia, or in industry – to practice.



SAE International AADL Standard Suite (AS-5506 series)

Core AADL language standard [V1 2004, V2 2012, V2.2 2017]

- Focused on *embedded software system modeling, analysis, and generation*
- Strongly typed language with well-defined semantics for execution of threads, processes on partitions and processor, sampled/queued communication, modes, end to end flows
- Textual and graphical notation
- V3 in progress: interface composition, system configuration, binding, type system unification

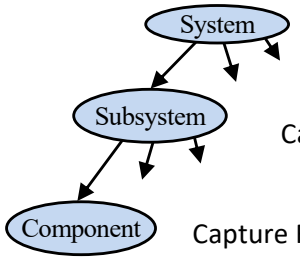
Ongoing work to align AADL and SysML in a common workflow -> Adventium Labs, ANSYS, SEI

Standardized AADL Annex Extensions	AADL Annexes in Progress
• Error Model language for safety, reliability, security analysis [2006, 2015] • ARINC653 extension for partitioned architectures [2011, 2015] • Behavior Specification Language for modes and interaction behavior [2011, 2017] • Data Modeling extension for interfacing with data models (UML, ASN.1, …) [2011] • AADL Runtime System & Code Generation [2006, 2015] • FACE Annex [2019]	• Network Specification Annex • Cyber Security Annex Roadmap • Requirements Definition and Assurance Annex

AADL Error Model Scope and Purpose

System safety process uses many individual methods and analyses, e.g.

- hazard analysis
- failure modes and effects analysis
- fault trees
- Markov processes



Capture hazards

Capture risk mitigation architecture

Capture FMEA model

Goal: a general facility for modeling fault/error/failure behaviors that can be used for several modeling and analysis activities.

Annotated architecture model permits checking for consistency and completeness between these various declarations.

Related analyses are also useful for other purposes, e.g.

- maintainability
- availability
- Integrity
- Security

SAE ARP 4761 *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*
Demonstrated in SAVI Wheel Braking System Example

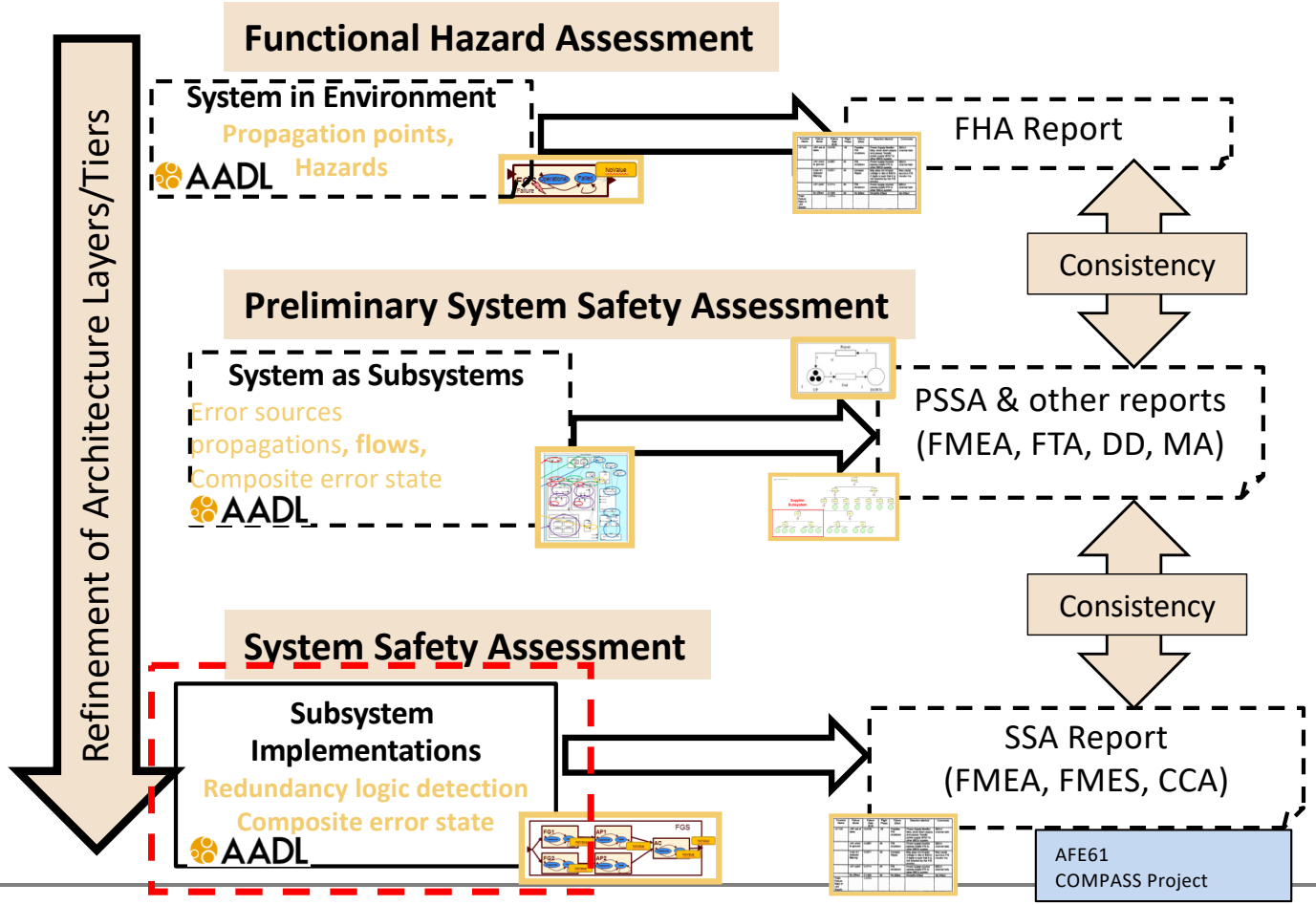
Error Model Annex can be adapted to other ADLs

Iterative Safety Analysis Process with AADL

Hazard analysis, system-specific

Early system designs, extending fault model

Safety guaranteed by generic architectural design patterns



Actionable safety patterns – workplan

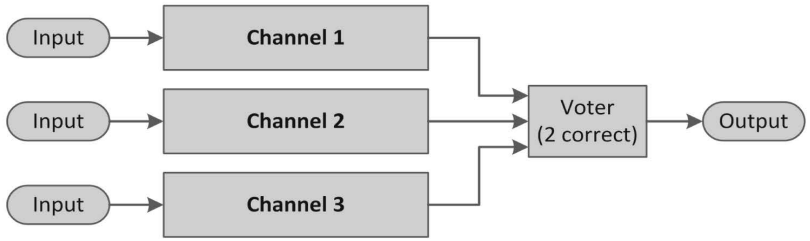
Safety & security patterns are dual-sided

- As *patterns*, they propose a collection of design artefacts that fulfill some high-level requirements like mitigation some errors, improving reliability or security
- As *tactics*, they propose *a* rationale for applying a pattern, and discuss improvements to the system, and eventually drawbacks/limitations

In both cases, those are informally designed, strong expertise required to

- Define them: what is the best way to convey a specific pattern definition?
- Select them: which one is adapted to the current system design?
- Apply them: how to weave existing architecture with new components?
- Combine them: applying patterns is not commutative. Is there a preferred order?

TMR pattern
Adapted from [1]

Pattern Name	Triple Modular Redundancy Pattern	Pattern Type	hardware, failover
AlsoKnownAs	2oo3 Pattern, Homogeneous Triplex Pattern		
Context	A safety-critical application without a fail-safe state, potentially many random and few systematic faults.		
Problem	How to design a system which continues operating even in the presence of a fault in one of the system components.		
Forces	- the system cannot shut down because it has no safe state - safety standard requires high fault coverage for single-point of failure components - high availability requires hardware platforms to be maintained at the runtime		
Solution	Three identical hardware channels operate in parallel. If a single fault occurs in one channel then the other two channels still produce the correct output. A majority voter decides for the correct result. 		

Imprecisions:

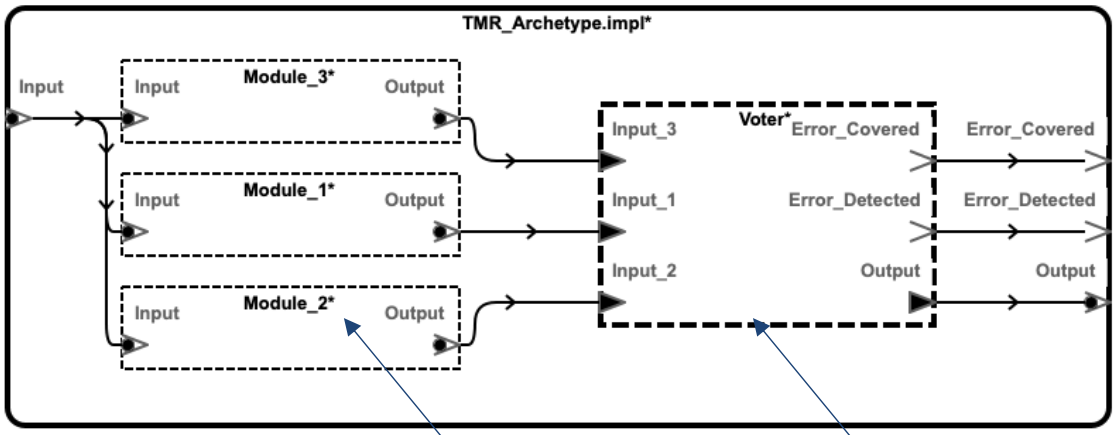
- "input" the same? or three different paths (sensors ?) to get inputs?
- Voter algorithm and tolerance?
- Synchronization on voter inputs

[1] C. Preschern et al. *"Safety Architecture Pattern System with Security Aspects"*, https://doi.org/10.1007/978-3-030-14291-9_2

From Patterns to Models – step #1 AADL models

Operationalize existing patterns:

- A pattern =
  - set of abstract components +
  - behavior +
  - fault model

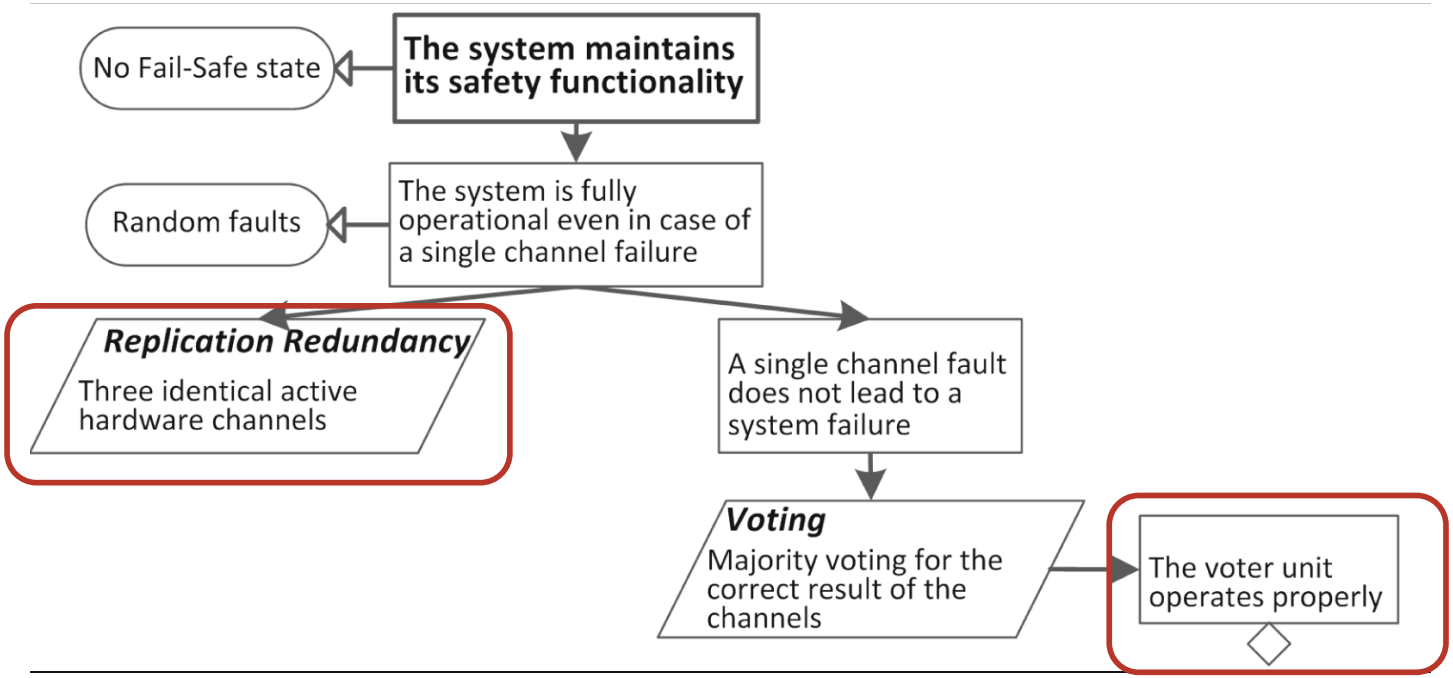


Faulty components (AADL/EMV2)

Voter (AADL/BA) + EMV2

TMR pattern
Adapted from [1]

Pattern is also defined by a Safety GSN, providing grounds for safety arguments



[1] C. Preschern et al. "*Safety Architecture Pattern System with Security Aspects*", https://doi.org/10.1007/978-3-030-14291-9_2

From Patterns to Models – step #2 ALISA verification plans

GSN acts as a template for ALISA verification plans, a companion DSL for AADL

- ALISA Goals are high-level concerns

```
goal goal_single_channel_failure [
   category Pattern_Goal.Safety
            Safety_Pattern_Context.Random_Faults
   description "The system is fully operational even in case of single channel failure"
   stakeholder Patterns_Role.Safety_Auditor
```

Category of goals

- ALISA Requirements are intermediate or terminal nodes

```
system requirements TMR_Requirements for TMR::TMR_Archetype.impl [
   description "High-level requirements for the TMR pattern."
   see goals TMR_Stakeholders_Goals

   requirement TMR_Feature_Output for Output
[.   category Pattern_Goal.Safety
     description "The voter unit operated properly"

     development stakeholder Patterns_Role.Safety_Architect
     see goal TMR_Stakeholders_Goals.goal_single_channel_failure_2
```

Link to component instance

Link to specific model element

From Patterns to Models – step #2 ALISA verification plans

- ALISA requirements are attached to claims that are bound to verification methods

```
verification plan TMR_Plan for TMR_Requirements [
   claim TMR_Requirements.TMR_Replication_Redundancy [
   activities
           checkReplicas : TMR_Registry.Three_Replicas ( module )

           check_Module_1 : AwasMethods.isQueryNotEmpty(q4)
           check_Module_2 : AwasMethods.isQueryNotEmpty(q5)
           check_Module_3 : AwasMethods.isQueryNotEmpty(q6)

           no_common_sources_1 : AwasMethods.isQueryEmpty(q1)
           no_common_sources_2 : AwasMethods.isQueryEmpty(q2)
           no_common_sources_3 : AwasMethods.isQueryEmpty(q3)
   ]

   claim TMR_Requirements.TMR_Voter [
   activities
           AGREE_Check: TMR_Registry.Voter_Correct ()
   ]
```

Claim this requirement is verified

Execution of analysis from OSATE or 3rd party plugins

About “Replication Redundancy” goal

“Three identical HW channels” is incomplete

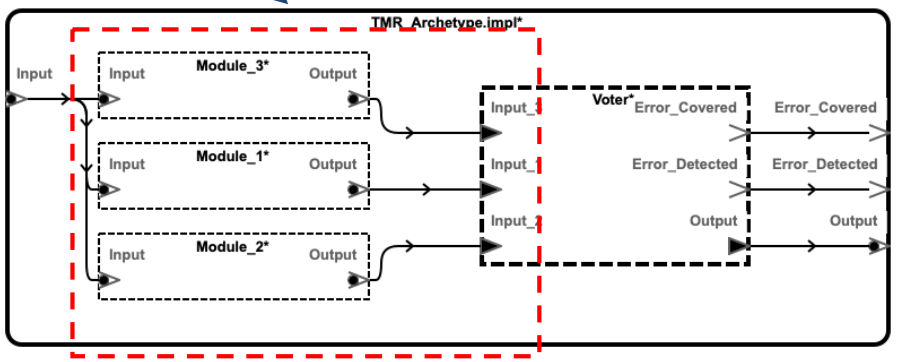
- They can by dissimilar but provide the same data/service
- Must be ultimately three disjoint channels proposing the same data

Properties on topology

- For all i, Input_i is connected to a component implementing Module
- Backward path to Input_i does not intersect backward path to Input_j for i /= j

⇒ Use AWAS by Kansas State University

Global composition
(tool: AWAS)

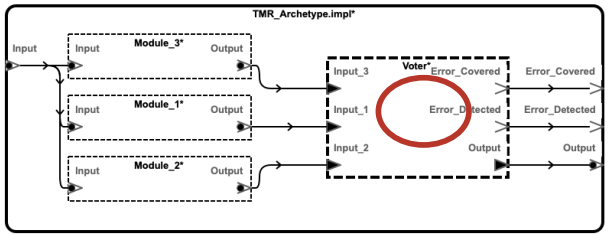



```
// AWAS Query (subset)
val q1 = " (reach backward Voter.Input_3)
 intersect (reach backward Voter.Input_1)"
```

About ”The voter unit operates properly” goal 1/2

Use AADL/BA for component behavior

- Compatible with code generation



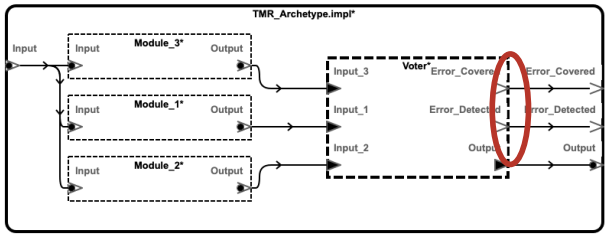

```
abstract implementation Voter.impl
annex behavior_specification {**
states
   idle    :  complete final state;
   compute : initial state;
transitions
   t1 : compute -[Input_1? = Input_2? and Input_1? = Input_3?]-> idle {Output := Input_1?};
   --  [..]
   t5 : compute -[Input_1? != Input_2? and Input_1? != Input_3?]
      -> idle {Output := Input_1?; Error_Detected!};
**};
annex agree {**
-- ..
**};
end Voter.impl;
```

Implementation

About ”The voter unit operates properly” goal 2/2

Use AGREE by Collins for interface contract

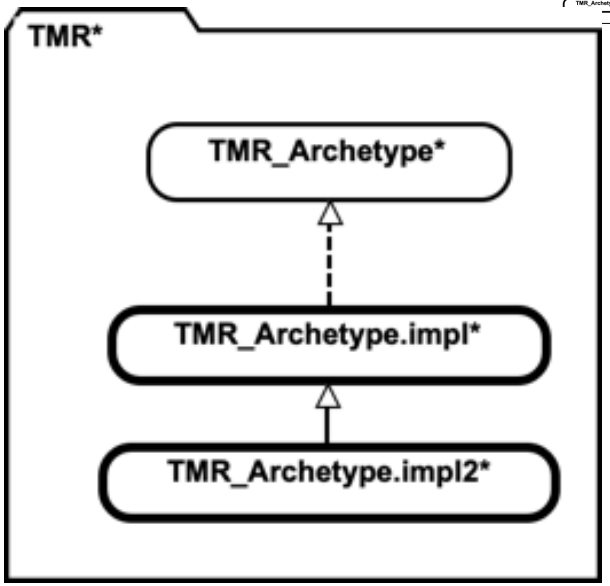
- Enables model checking
- Compliance between contract and BA




```
abstract implementation Voter.impl
annex behavior_specification {**
-- ..
**};
annex agree {**
  eq output_eq : int = Input_1 -> if ( (Input_1 = Input_2) and (Input_1 = Input_3))
        then Input_1 else pre(output_eq);
  eq error_detected_eq : bool = false -> (Input_1 <> Input_2) and (Input_1 <> Input_3);
  --  Map equations to BA outputs
  assert(Output = output_eq);
  assert(Error_Detected = error_detected_eq);
**};
end Voter.impl;
```

Contract

TMR pattern application



Option#1: Extension/refinement of provided pattern template

⇒ Direct application of AADL extends/refine mechanisms

But implies the system architecture matches the pattern

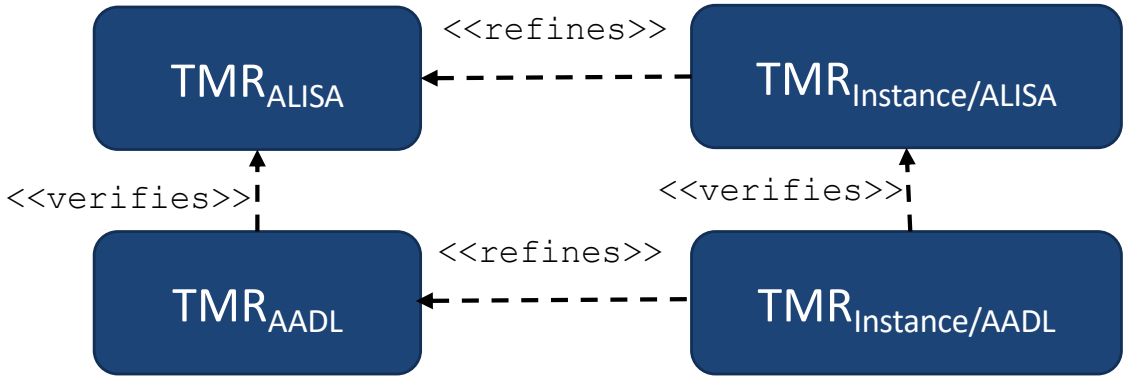
- Not applicable if one want dissimilar input modules
- Not applicable if voting is a thread of a larger software process (e.g. breaking of the hierarchy).

TMR pattern application

Option#2: Consider Safety pattern as "dual-layered patterns"

- Architectural patterns: roles, and data flows
- Verification plan patterns: abstract verification objectives

An instance of the pattern refines both layers to match a specific problem



Conclusion

Safety & security patterns form the foundation for rigorous safety-critical engineering

- Many patterns exist, but no reusable library of patterns
- Limit applicability, error-prone in complex design

Contributions

- Apply AADL and AADL extensions (AGREE, ALISA, AWAS) propose a systematic definition of safety patterns: goals, requirements, verification methods and AADL abstract models and
- a process to apply patterns to specific architecture

This library will be integrated in a future release of OSATE.