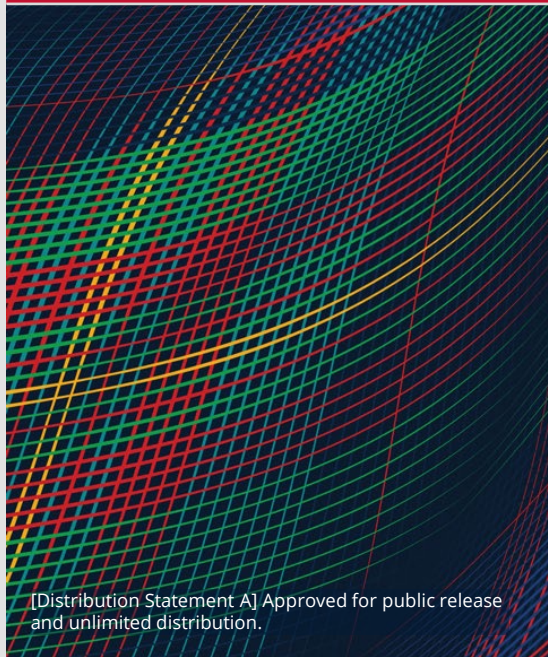
Model-Based Contracts and Argumentations for Assuring Safety-Critical Systems

AUGUST 21, 2025

Dionisio de Niz

Carnegie Mellon University
Software Engineering Institute

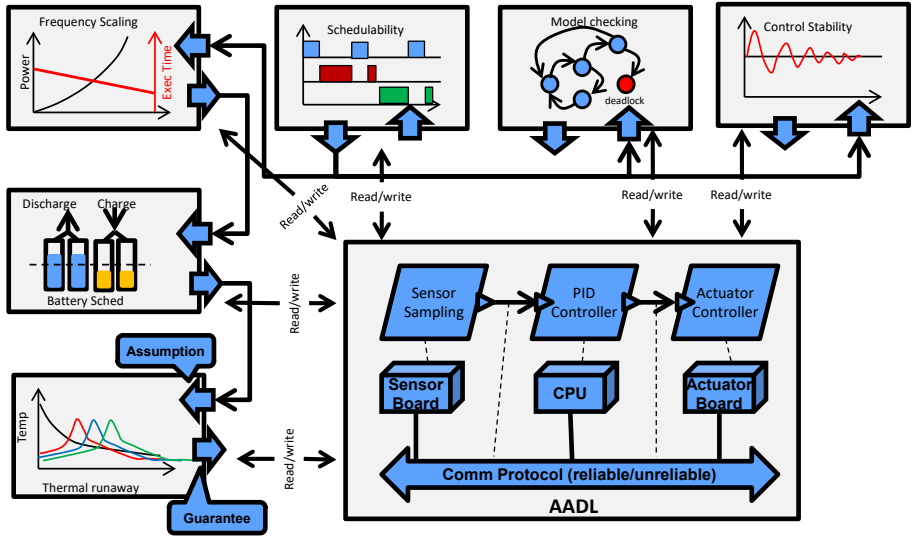


Model-Based Engineering: Multiple Claims – Multiple Analyses

Different Assurance Claims

- Combine multiple analysis
- Validate assumptions
- Resolve assumption conflicts

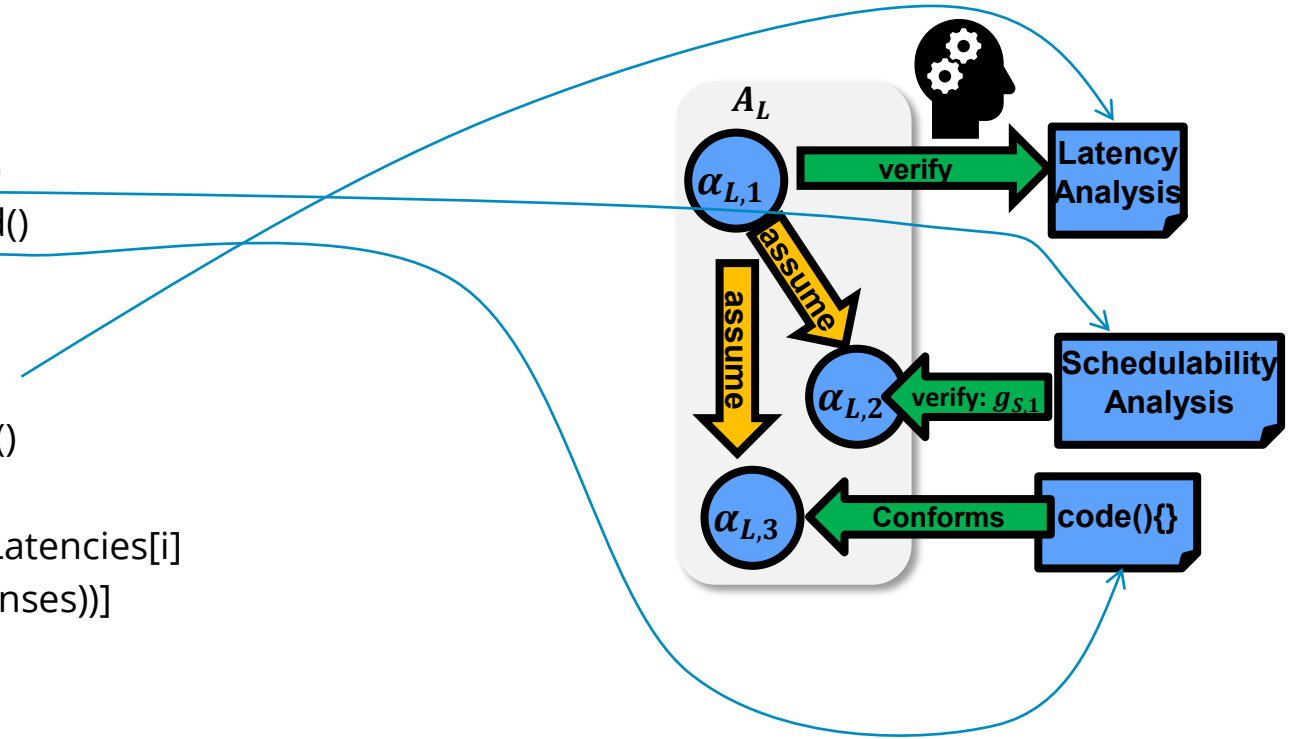
Integrate into arguments to satisfy claims



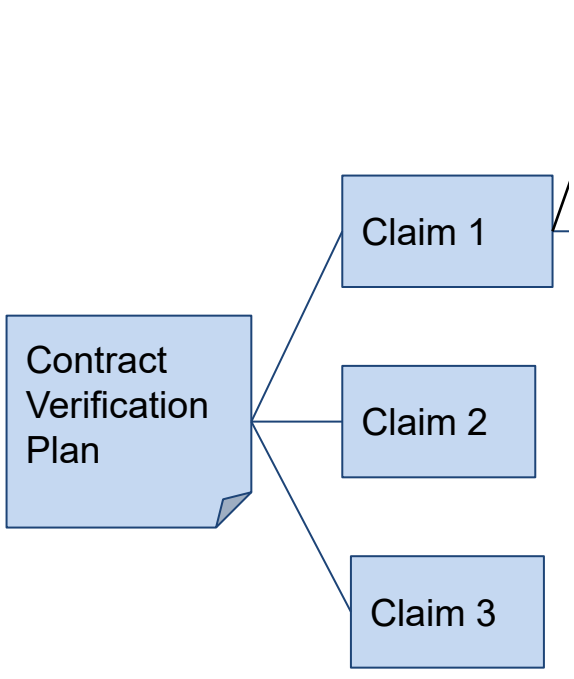
Analysis Contract: Tracking Assumptions and Guarantees

contract {
  inputs:
    E2ELatancies
  assumptions:
    areConnectionsDelayed()
    areDeadlinesConstrained()
    areTasksSchedulable()
    areAllThreadsPeriodic()
  analysis:
    meetEndToEndLatencies()
  guarantee:
    [E2EResponses[i] <= E2ELatencies[i]
      for i in range(len(Responses))]
}

$C_L = (A_L, G_L)$ with $A_L = \{\alpha_{L,1}, \alpha_{L,2}, \alpha_{L,3}\}$

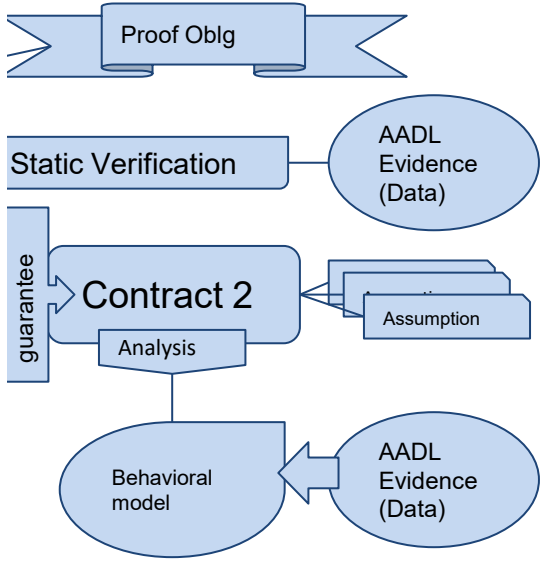


Assurance Contract Argumentation




```
1  annex contract {**
2      contract <name> {
3      queries
4          <model var> =
5              <query to obtain model data>
6      domains
7          <domain reference>
8      input assumptions
9          <Bool func to check data consistency>(
10             <model vars>)
11     assumptions
12         <Bool func>(<model vars>)
13         –> <symbolic assertion>
14     analysis
15         <Bool func>(<model vars>)
16         –> <symbolic guarantee>
17     }
18 **};
```

Listing 1: Contract Template



Symbolic Contract Argumentation
Symbolic Assurance Refinement Framework

Assumptions

- Constraints that must be satisfied for a valid analysis

Analysis

- Evaluate whether the guarantee can be discharged

Guarantee

- Assertion presented as a true fact on model

Implementation

- Satisfiability Modulo Theories (Z3)
- Implements contract argumentation
  - Evaluate whether constrains can be satisfied with facts from analysis guarantees
- Validate assumptions
  - Proof obligations: lack of constraints allow any value that satisfy assumption (e.g., RM priorities)

Refinement & Analysis Contract Selection

Argumentation evolves with model refinement

- Unconstrained symbolic variables assigned value to satisfy problem
- Considered as proof obligations (due to partial model)

Verify final complete model for all possible values

Analysis Contract Selection

Refinement-Based Selection

- Select contract based on model data availability

Design-Based Selection

- Select contract based on model data type (e.g., RM priorities)

Contract Argumentation Scalability

Exploit Knowledge from Scientific Domain

- Efficient algorithms from specialized domains
  - E.g., greedy worst-case response time in real-time theory
  - Implemented in imperative languages

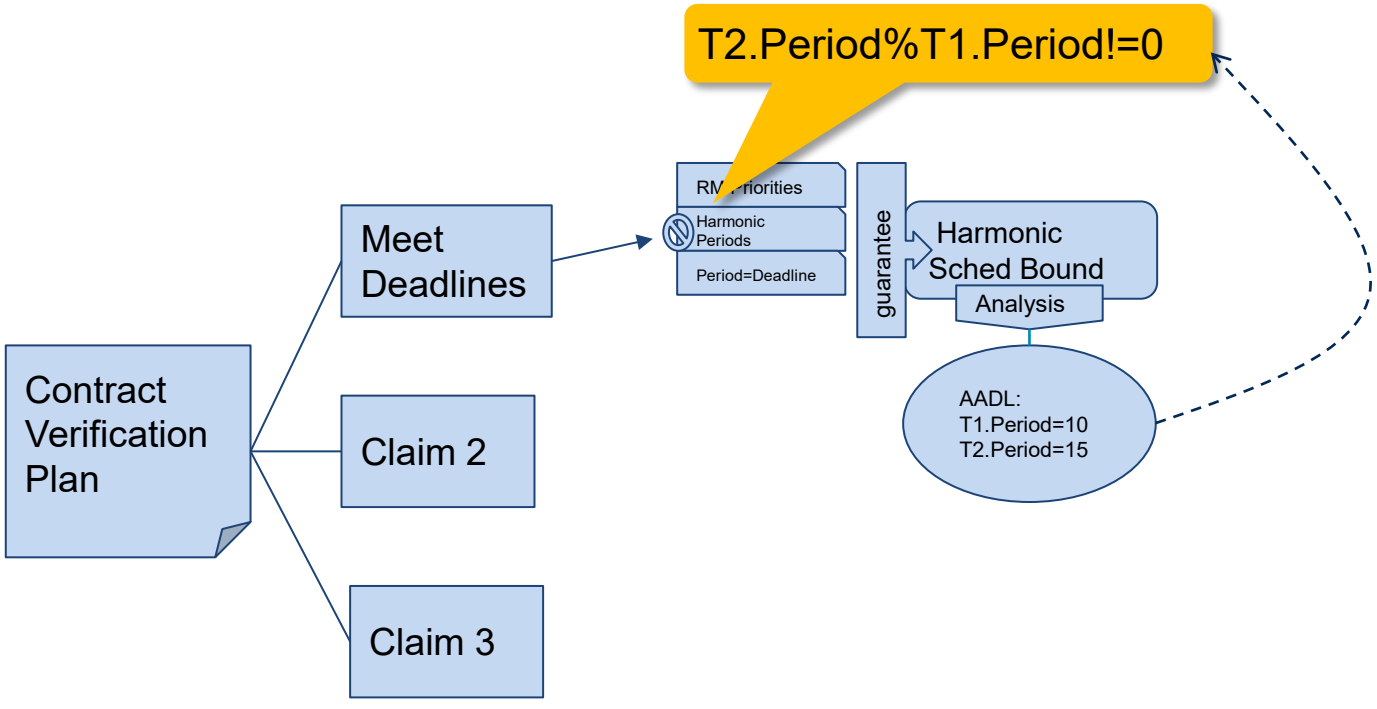
Assume correctness of analysis

- When validating the contract argumentation
- To be connected with other lower-level verification results
  - E.g., PROSA: coq (theorem prover) verification of real-time theory

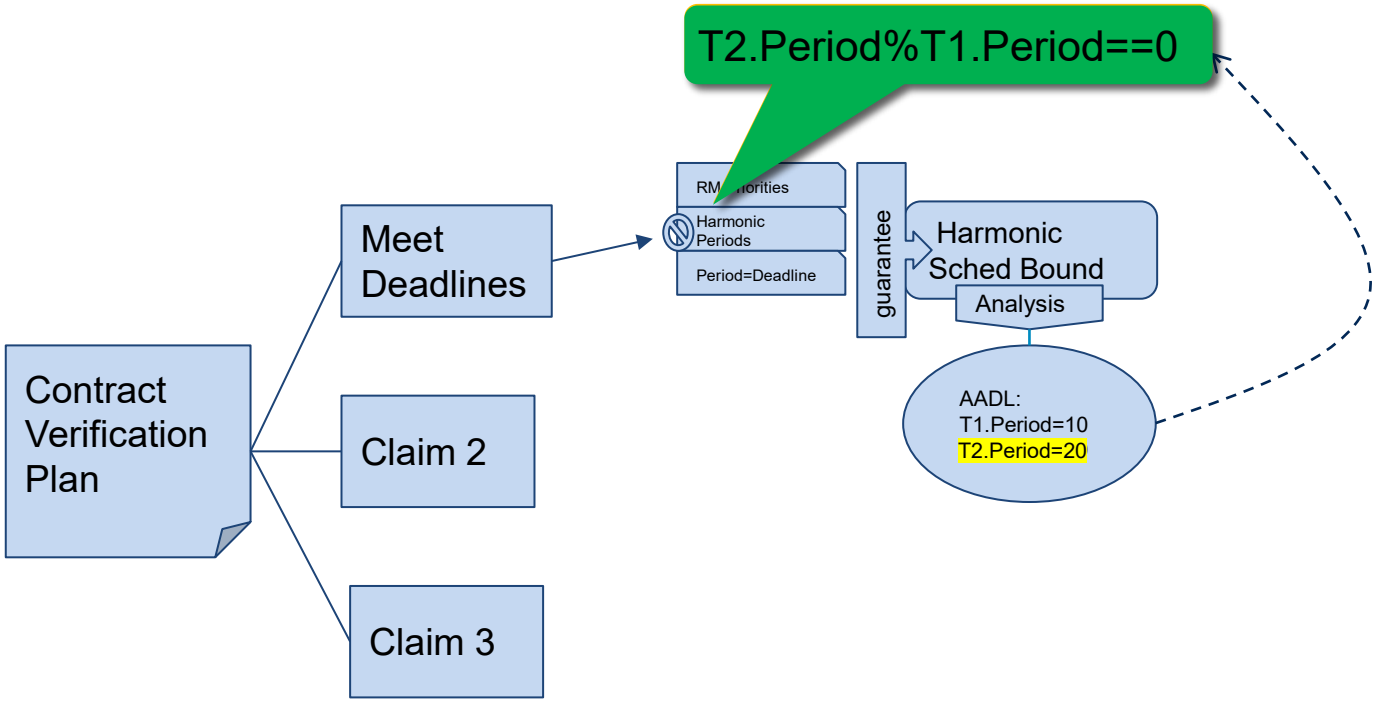
Correctness of implementation

- Exploit proven properties of runtime mechanisms: e.g., schedulers, hypervisors
- Exploit code generation
- Deferred code verification to conform to assumptions

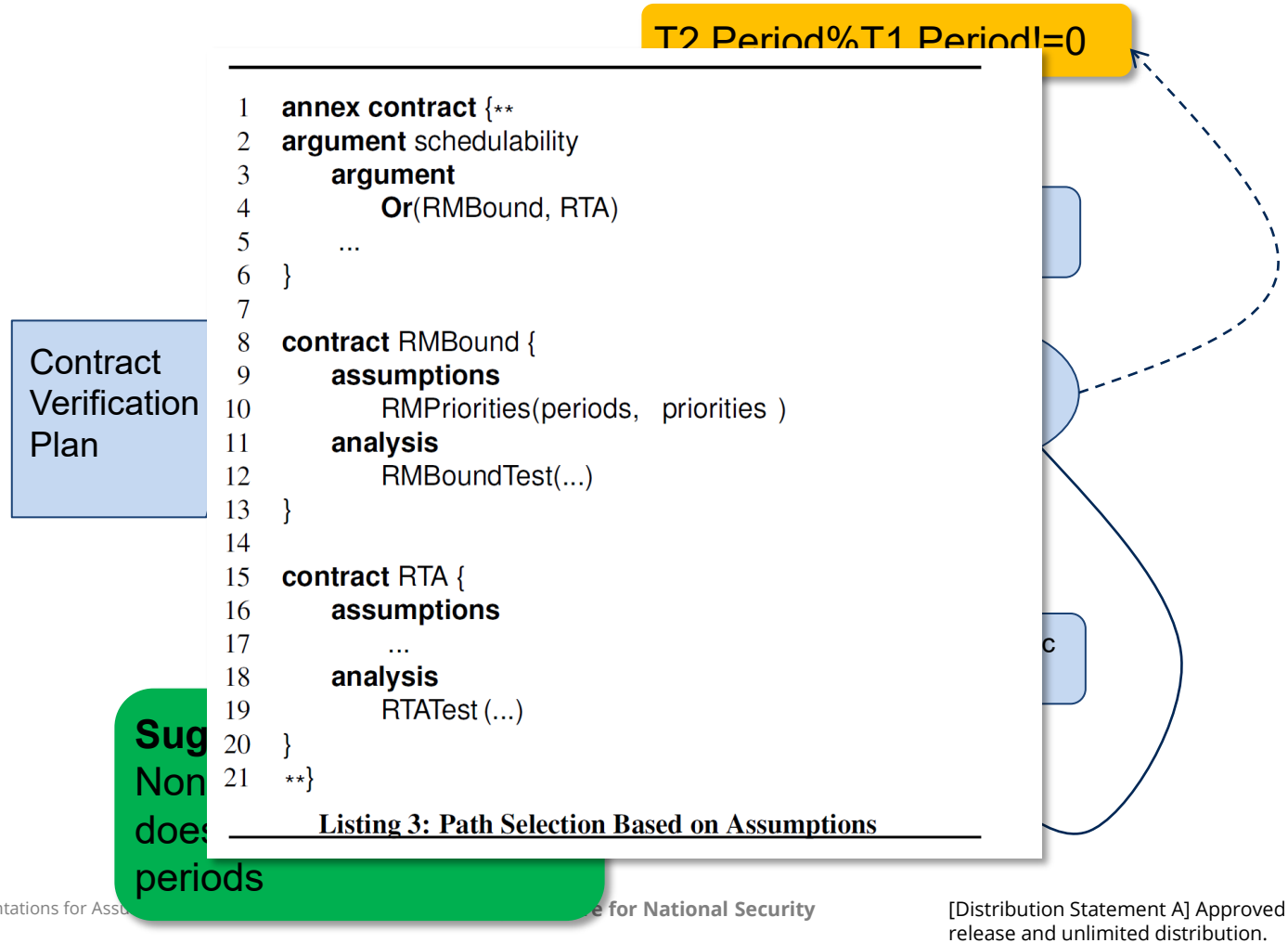
Repairing Assumptions



Repairing Assumptions



Repairing Assumptions




```
1   annex contract {**
2   argument schedulability
3       argument
4           Or(RMBound, RTA)
5       ...
6   }
7
8   contract RMBound {
9       assumptions
10          RMPriorities(periods,   priorities )
11      analysis
12          RMBoundTest(...)
13  }
14
15  contract RTA {
16      assumptions
17          ...
18      analysis
19          RTATest (...)
20  }
21  **}
```

Listing 3: Path Selection Based on Assumptions

Argument Modularity

Decomposed into subclaims

Contract Verification Plan

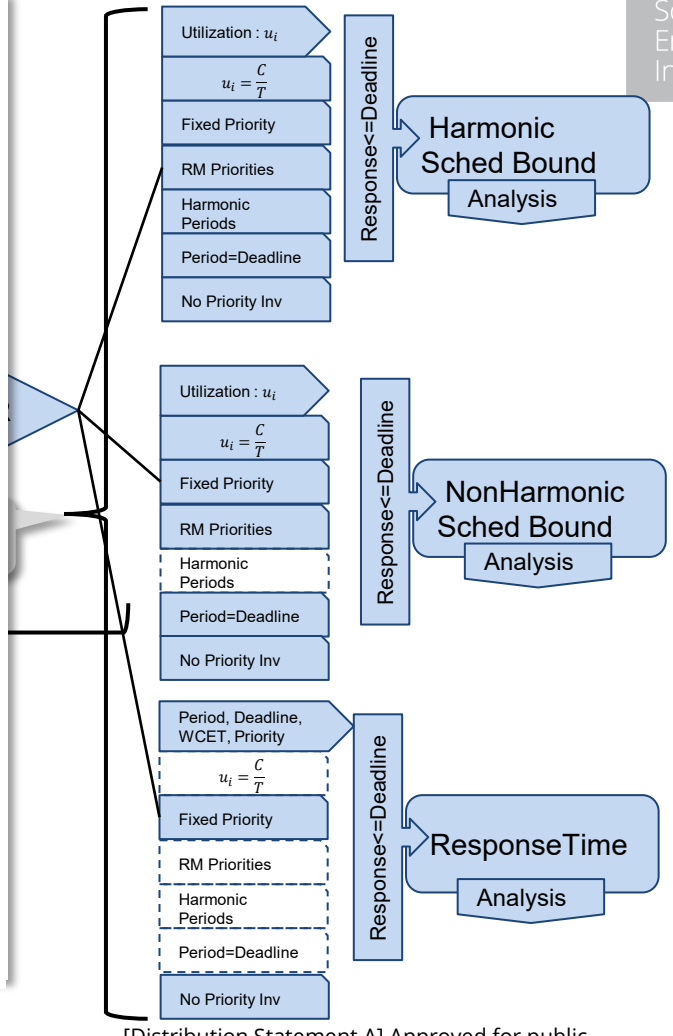
Reliable Autobrak

Claim 2

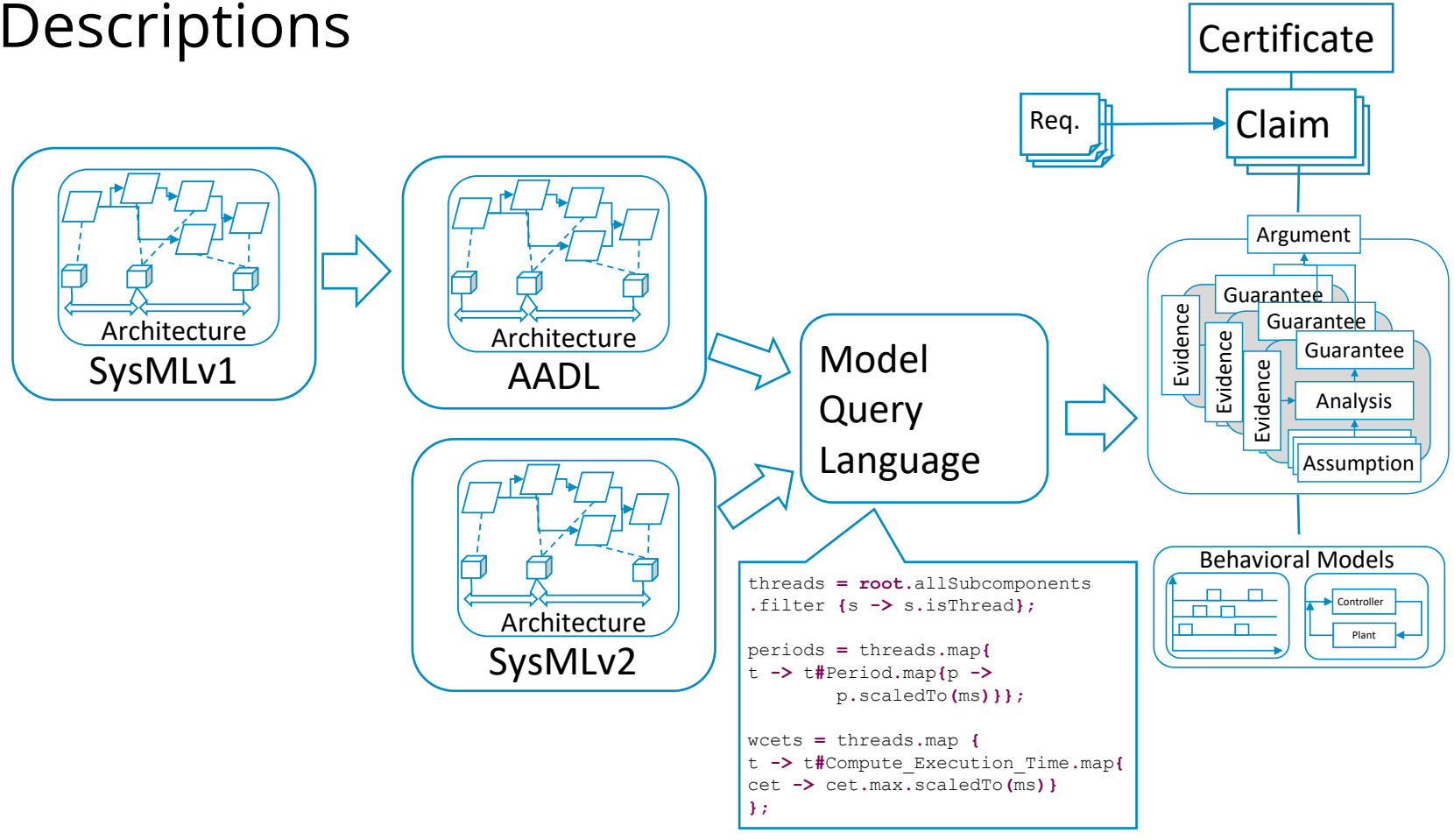
Claim 3

```
1  annex contract {**
2  verification plan myPlan {
3      claims
4          EndToEndDelayArgument–>
5          And([E2EResp[i] <= E2ELatency[i]
6                  for i in range(len(E2EResp))]
7  }
8
9  argument EndToEndDelayArgument {
10     argument
11         Or(E2ESched, E2ESFlowSpec)–>
12             And([E2EResp[i] <= E2ELatency[i]
13                     for i in range(len(E2EResp))]
14 }
15
16 contract E2ESched {
17     input assumptions
18         allSchedDataPresent()
19     ...
20 }
21
22 contract E2EFlowSpec {
23     input assumptions
24         notAllSchedDataPresent()
25     ...
26 }
27 **}
```

Listing 2: Path Selection Based on Refinement



Analytic Argumentation from Multiple Architectural Descriptions

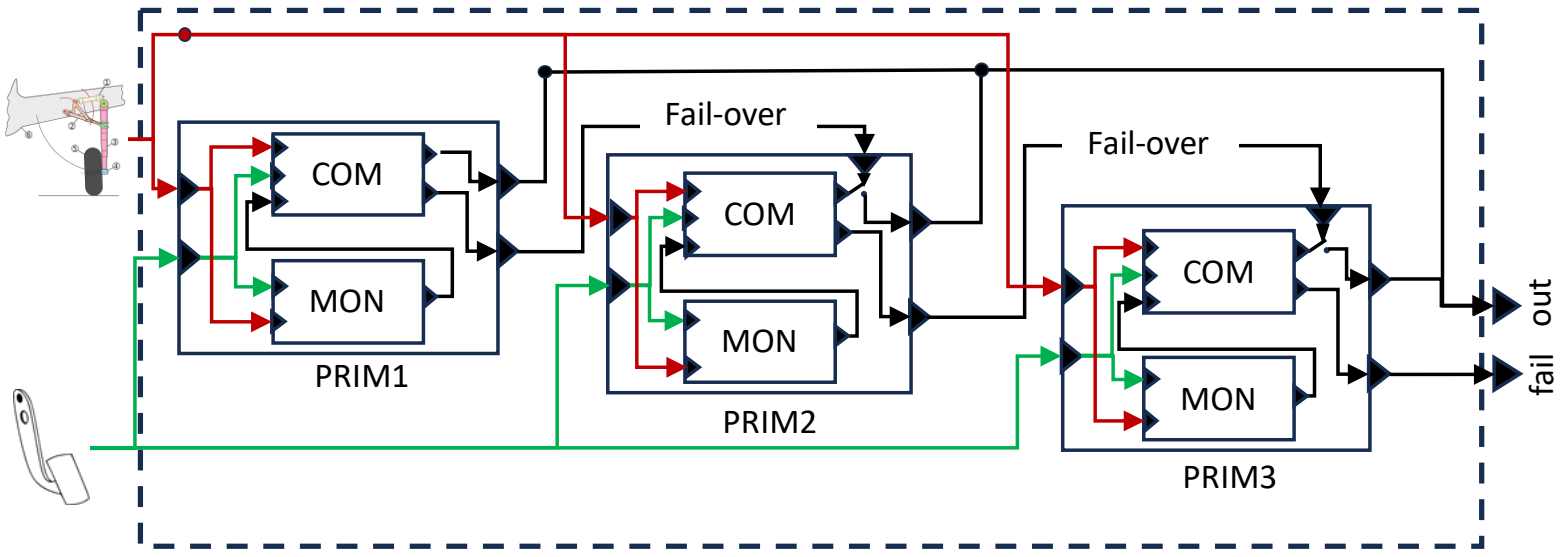


Flight CI202 Incident in On June 14, 2020

Incident Report:

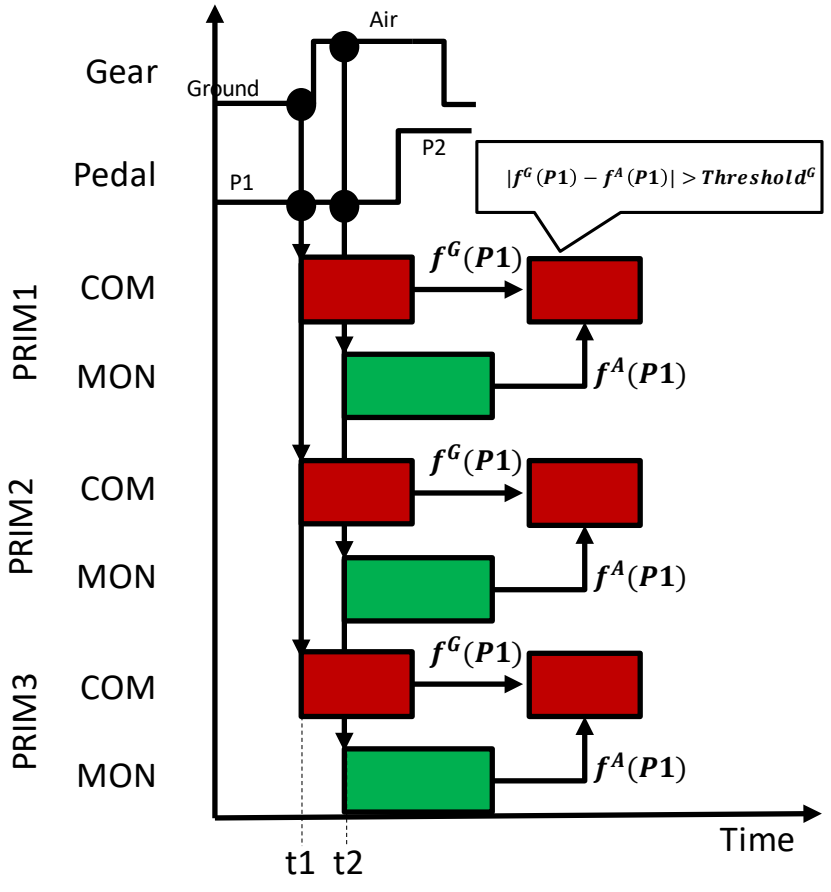
- Aircraft: Airbus A330-302
- At touchdown in Taipei quasi-simultaneous failure of the three flight-control primary computers (FCPC or PRIM)
- Ground spoilers, thrust reversers, and autobrake were lost
- Flight crew:
  - aware of the autobrake and reversers failure to activate
  - applied full manual brake rapidly:
    safely stopped aircraft 30 feet before the end of runway

Partial View of A330 Architecture (interpreted from report)



Drawing from public domain: https://commons.wikimedia.org/wiki/File:Landing_gear_schematic_colored.svg

Channel Asynchronism



Two Replication Strategies +Modes

Integrity (computing errors - bugs)

- Replicas: COM+MON
  - Same outputs (cross checked)
- Assumption: same inputs

Availability (hardware failure)

- Replicas: PRIM1 + PRIM2 + PRIM3
  - Output even if failures
- Assumption: failure independence

A330 Analysis Assumptions

Integrity

- Different teams develop modules independently
- Module pair should use the same input

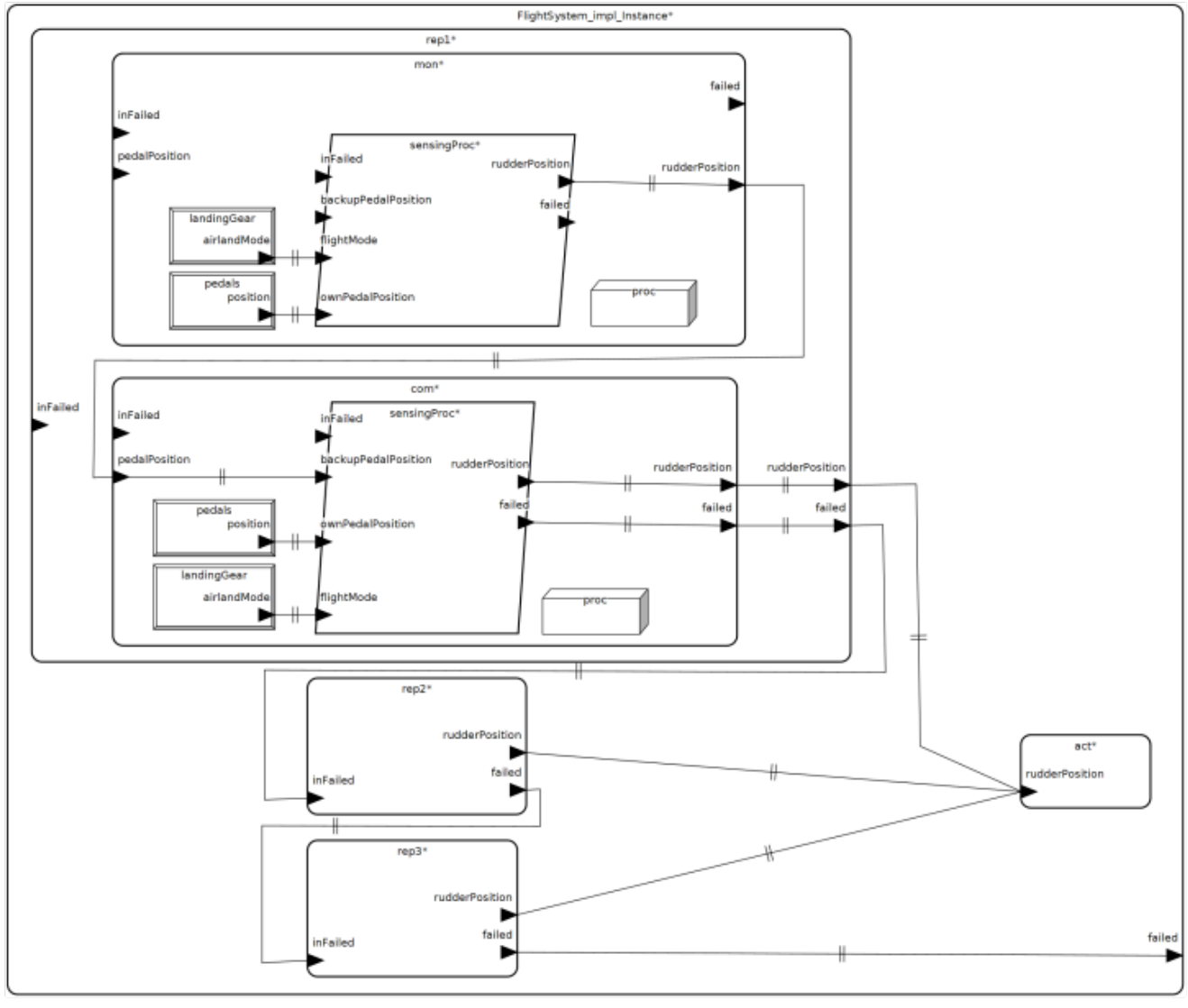
Availability

- Modules must fail independently
- Analysis: FTA

End-To-End Timing

- Communication Style: read output from sender's previous cycle
- RM priorities
- No Self Suspension

Original Design

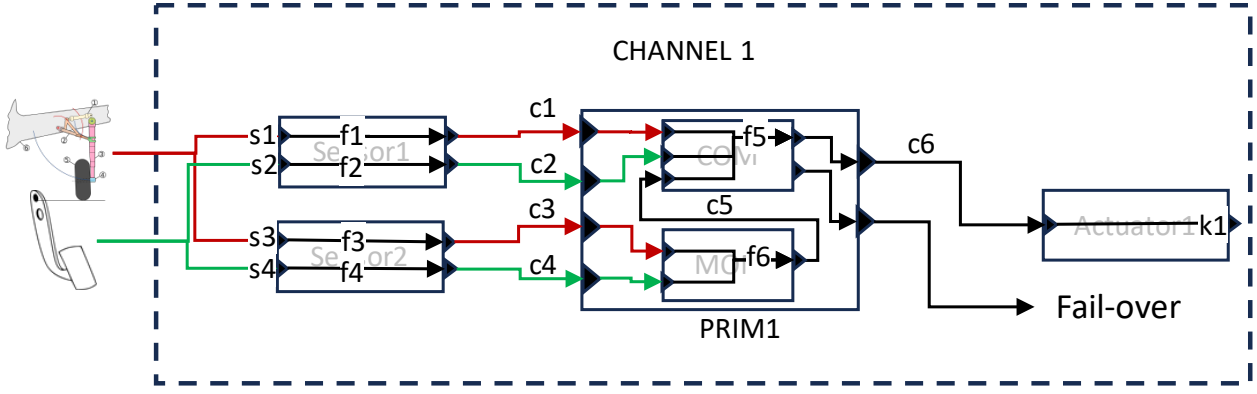


Channel Modeling

End-to-End Flow in AADL

```
pedalToActuationCOMPRIM1: end to end flow Sensor1.s1->
Sensor1.f1->c2->COM.f5->c6->Actuator1.k1;

pedalToActuationMONPRIM1: end to end flow Sensor2.s3->
Sensor2.f3->c4->MON.f6->c5->COM.f5->c6->Actuator1.k1;
```



Availability Replication

Identifying Replicas

```
ReplicationProperties::Replicating =>(reference
(pedalToActuationMONPRIM2), reference
(pedalToActuationMONPRIM3)) applies to pedalToActuatorMONPRIM1;
```

Specifying Availability Properties and Goals

```
ReplicationProperties::ReliabilityTarget => 0.85 applies to
pedalToActuationMONPRIM1;

ReplicationProperties::FailureProbability => 0.01 applies to
Sensor1.processor;
```

Integrity Replication

Identifying Integrity Replicas

```
ReplicationProperties::IntegrityReplicas =>(reference
pedalToActuationCOMPRIM1) applies to pedalToActuationMONPRIM1;
```

Specifying Jitter Tolerance

```
ReplicationProperties::ReplicasStartJitterTolerance => 0 ms
applies to pedalToActuationCOMPRIM1;
```

Verification Plan and Arguments

```
verification plan verifySynchronization {
     component
          s: EndToEndTimingExample::mysystem.i;
     domains
          synchronization;
          reliability;
     claims
          `And([E2ESamplingJitter[i] <=
               E2ESamplingJitterTolerance[i]
                    for i in range(len(E2ESamplingJitter))])`;
          `And([Reliability[i]>=ReliabilityTarget[i]
                    for i in range(len(Reliability))])`;
          `And([E2EResponses[i] <= E2ELatencies[i]
                    for i in range(len(E2EResponses))` ;
     contracts
          SamplingSynchronizationContract;
          EndToEndDelayedCommunicationContract;
          ReliabilityContract;
}
```

Sampling Jitter Contract

```
contract SamplingSynchronizationContract {
     domains
          synchronization;
     guarantee
          <=> `And([E2ESamplingJitter[i] <=
                E2ESamplingJitterTolerance[i]
                for i in range(len(E2ESamplingJitter))])`;
     analysis
      '''areFlowsInSync1(${synchronization::flowComponents$}
         ,error0)''';
}
```

Reliability Contract

```
contract ReliabilityContract {
        domains
                reliability ;
         assumptions
                '''areReplicasOnIndependentProcessors(
                    ${synchronization::flowComponents$},
                    error0)''';
        guarantee
                ⇔ `And([Reliability[i] >= ReliabilityTarget[i]
                    for i in range(len(Reliability))` ;
        analysis
                '''isE2EFlowProbabilityOfFailureMet(
                    ${replicatede2es$},error0)` ;
   }
```

Timing Contract


```
contract EndToEndDelayedCommunicationContract {
	domains
		schedulability;
	queries
	input assumptions
		'''areEndToEndLatenciesInputDataComplete(
			${periods$}, ${wcets$}, ${deadlines$}, ${names$})''';
	assumptions
		contract areConnectionsDelayedContract;
		'''areAllThreadsPeriodic(${threads$}, ${prot$},${names$},error0)'''
			=> `And([Periodics[i] for i in range(len(Periodics))])`;
		'''areAllDeadlinesConstrained(
			${threads$},${periods$},
			${deadlines$}, ${names$},error0)'''
			=> `And([Deadlines[i] <= Periods[i]
				for i in range(len(Deadlines))])`;
		argument schedulabilityArgument;
	guarantee
		<=> `And([E2EResponses[i] <= E2ELatencies[i]
			for i in range(len(E2EResponses))])`;
	analysis
		'''meetEndToEndLatencies(${synchronization::flowComponents$},error0)''';
}
```

Original Architecture Analysis

Input Jitter of Zero Cannot be Achieved

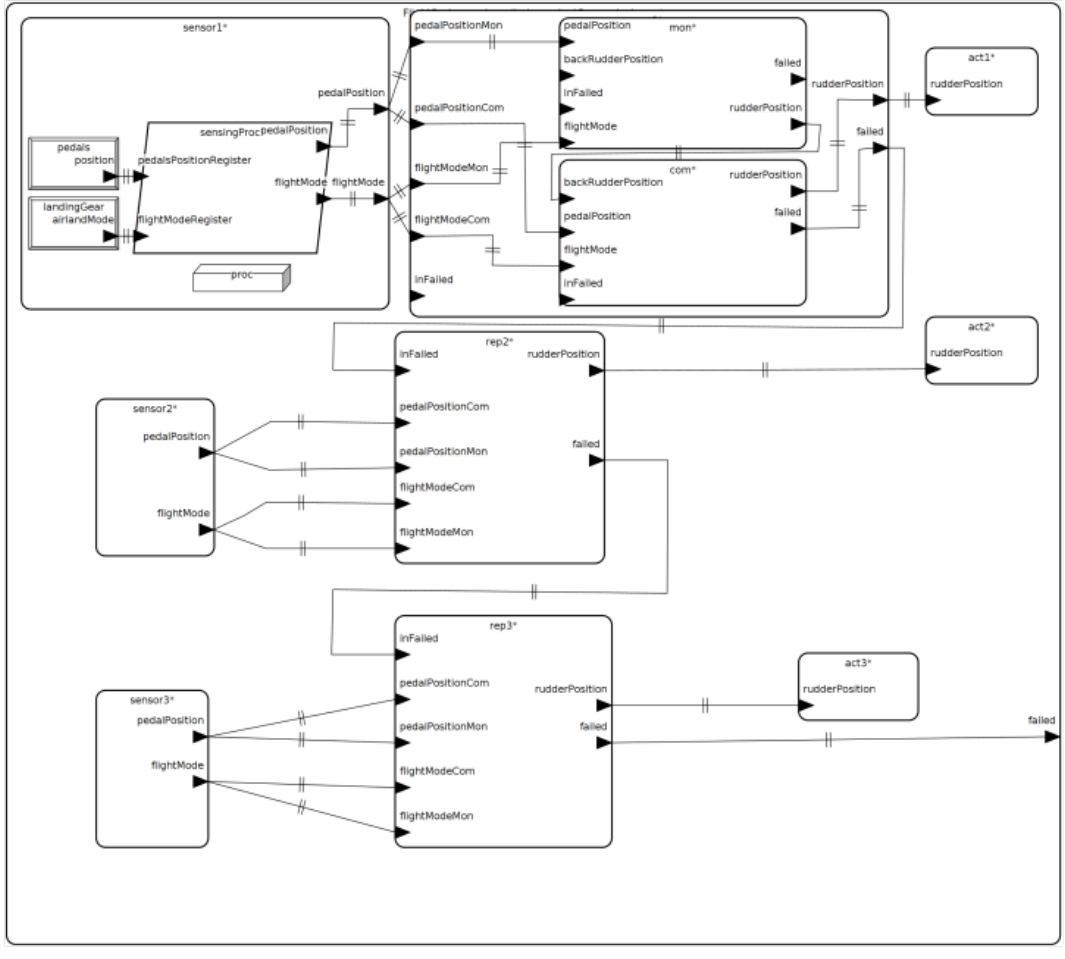
- Unless sophisticated (unrealistic) synchronization

COM and MON fail together

- If output disagreement

Common COM+MON sensing preserves diverse computation

Common COM+MON Sensor Details



Concluding Remarks

New Architecture

- Meet all claims
- Meet all assumptions

- Encoded in Formalized verification argumentation

Website:
https://acps-sei.github.io/tools/sar