

RESEARCH REVIEW 2022

Safety Analysis and Fault Detection Isolation and Recovery (SAFIR) Synthesis for Time-Sensitive Cyber- Physical Systems

NOVEMBER 15, 2022

Jerome Hugues and Keaton Hanna

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.
©2022

**Carnegie
Mellon
University**
Software
Engineering
Institute

The AI Fresh Breeze – Coming from an Iceberg

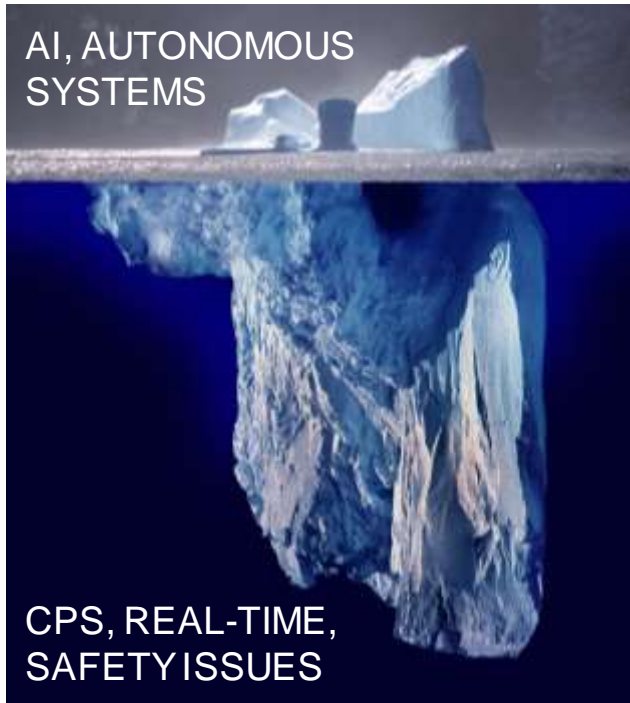


Image: CC BY-SA 3.0, Uwe Kils & Wiska Bodo,
[File:Iceberg.jpg - Wikimedia Commons, 2005](https://commons.wikimedia.org/wiki/File:Iceberg.jpg).

There is general enthusiasm for Increasingly-Autonomous CPS [1] to improve system efficiency (decrease # of operators), system capability (automate high-level tasks), and faster than human action.

Increasingly-Autonomous Systems embed advanced “intelligent” capabilities, from basic control to advanced AI.

But the fast pace of action and poorly-defined safety mechanisms make it impossible for a human to mitigate issues.

- ⇒ Distrust in system, longer V&V, or capability not deployed
- ⇒ May jeopardize capabilities of future DoD projects

[1] E. E. Alves, B. Devesh, B. Hall, K. Driscoll, A. Murugesan, and J. Rushby. Considerations in Assuring Safety of Increasingly Autonomous Systems. Technical Report NASA/CR-2018-220080, NF1676L-30426, NASA AIR TRANSPORTATION AND SAFETY, 2018.

The AI Fresh Breeze – Coming from an Iceberg



SAFIR aims to

- Deliver advanced Safety Analysis techniques to
 - Implement Fault Detection, Isolation, and Recovery policies for time-sensitive IA-CPS

There is general enthusiasm for Increasingly-Autonomous CPS [1] to improve system efficiency (decrease # of operators), system capability (automate high-level tasks), and faster than human action.

⇒ May jeopardize capabilities of future DoD projects

Image: CC BY-SA 3.0, Uwe Kils & Wiska Bodo,
[File:Iceberg.jpg - Wikimedia Commons, 2005](https://commons.wikimedia.org/wiki/File:Iceberg.jpg).

[1] E. E. Alves, B. Devesh, B. Hall, K. Driscoll, A. Murugesan, and J. Rushby. Considerations in Assuring Safety of Increasingly Autonomous Systems. Technical Report NASA/CR-2018-220080, NF1676L-30426, NASA AIR TRANSPORTATION AND SAFETY, 2018.

Case Study – UAV Patrolling



Let us consider a patrolling mission with the following:

Need to find and detect intruders, recognize threats

Partially known place: a factory, with slow-moving parts

Both closed and open areas; wind, lighting conditions

Tight maneuvers to enter/exit buildings, safety margins to avoid damages

Autonomy in decision making and man-machine teaming

Research questions:

How to guarantee that the system is safe to operate *and* will operate safely?

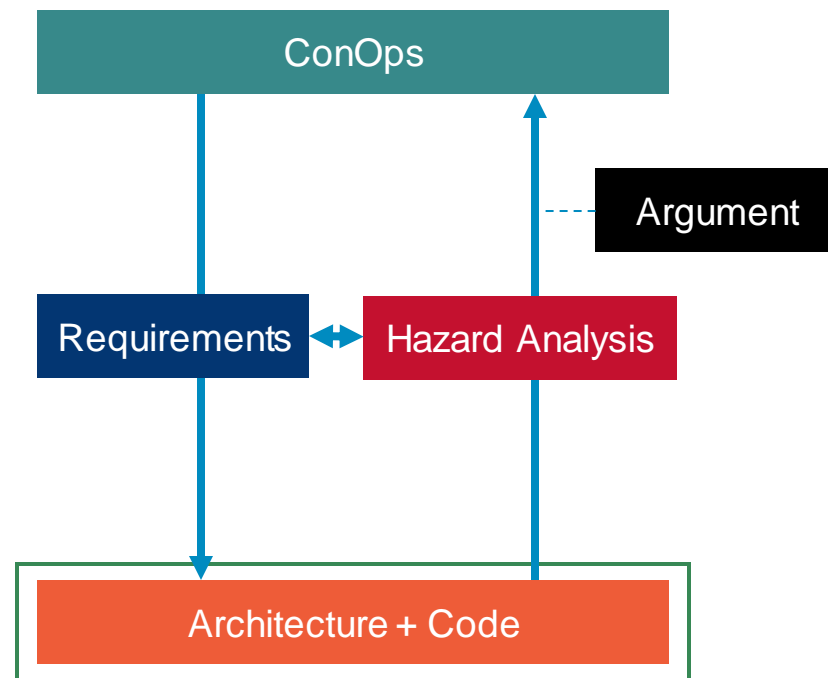
LSI SAFIR “Big Picture”

This **autonomous CPS_{context}** is safe because...

- Design time {
- It does **_____reqs**; it is implemented by **_____arch+code**
 - V&V **_____activities** demonstrate strict conformance
- Run time {
- It is operated **safely, and hazards or threats** are monitored and mitigated by **_____FDIR**.

LSI SAFIR is building a comprehensive approach to support Model-Based Systems Engineering and Safety Assessment through

- Architectural patterns – static and dynamic
- Tool-support analysis capabilities
- Argumentation to articulate all artifacts



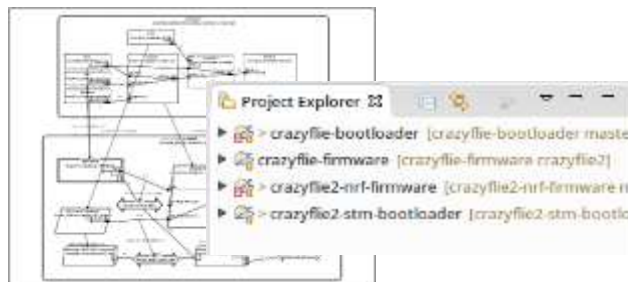
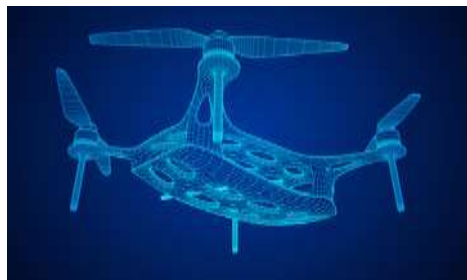
LSI SAFIR Core Contributions

SAFIR focuses on the engineering of safety-critical IA-CPS at the architectural level; i.e., it

Assumes operational hazards, sensor/actuators faults or vulnerabilities, timing anomalies, and AI functions misbehaviors are known

Guarantees the architecture properly mitigates faults down to the implementation through

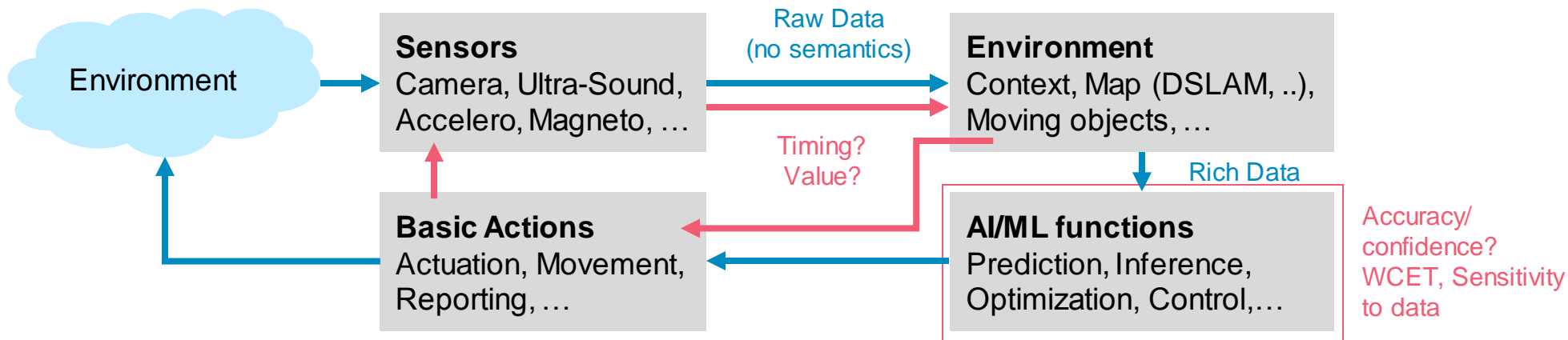
- Fault taxonomy, guidelines for selecting fault detectors
- Mechanized semantics of architectural description
- Representation of safety argumentation for review by certification authorities



LSI SAFIR Contributions

- 1. Fault Detection, Isolation, and Recovery (FDIR) provides the foundation to safety**
 - ⇒ **fault/attack detection mechanisms based on reinforcement learning**
 - ⇒ **FDIR patterns in the scope of autonomous systems**
 - ⇒ **reference architecture for FDIR-capable systems**
2. Mechanized semantics of architectural description
3. Generating arguments about system safety

Integrating AI Components – An Architecture Perspective



Dual challenge:

- Architecture -> AI/ML: Can the architecture mitigate erroneous inputs feeding LECs?
 - Fault detection + design guarantees
- AI/ML -> Architecture: How can we contain misbehaviors triggered by LECs?
 - Run-time assurance through fault detection, isolation, and recovery

AADL Standard Suite (AS-5506 series)

Core AADL language standard [V1 2004, V2 2012, V2.2 2017, V2.3 2022]

- Focused on embedded software system modeling, analysis, and generation
- Evidence produced as a result of automated tool-supported analysis
 - Performance analysis: worst-case response time, schedulability
 - Safety analysis: eliciting unsafe scenarios, computing fault trees, probability of reaching an unsafe state
 - Automated model review: conformance to modeling guidelines
 - Code generation: generating “correct-by-construction” software
 - Assurance process with ALISA

Standardized AADL Annex Extensions

- 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]

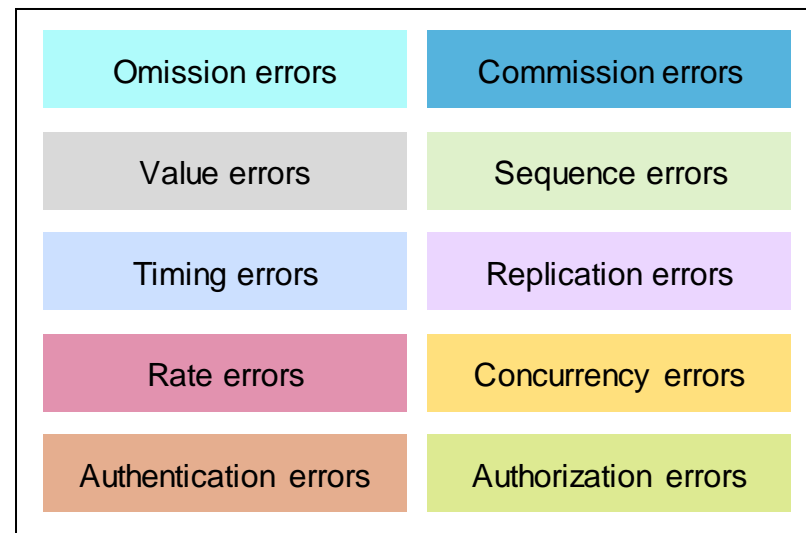
Fault Taxonomy Classification of Misbehaviors in IA-CPS

Notionally, the AI function receives data from the environment, takes a decision or acts on it.

Pragmatically, this data is processed by the CPS platform, inducing the following risks:

- Data tampering (no reference point) due to sensor faults, attacks, etc.
- Timing (blurring references) due to unavoidable latency in the system, timing violations or faults
- Pre-existing faults in non-AI CPS

Can be characterized using AADL EMV2 fault taxonomy



Fault Propagation Taxonomy

Part of SAE AADL Standard Suite

Contribution #1: Fault Detection

<https://doi.org/10.2514/6.2022-0969>, with GeorgiaTech

Research question: How to select a fault detector?

Solution: Define a decision procedure using the EMV2 fault taxonomy as pivot

Scenario :- set of errors it produces, $\text{error_set}\{\text{scenario}\}$

Detection :- set of errors it mitigates, $\text{error_set}\{\text{detector}\}$

A detector is efficient if

$$\text{error_set}\{\text{detector}\} \supseteq \text{error_set}\{\text{scenario}\}$$

⇒ **Contribution:** Survey of fault detectors, mapping of fault/attack scenario to detection mechanisms

Faults	Value			Timing		Presence		Quantity			Subtlety		Replication	
	H	L	U	E	L	C	O	SI	S	Sv	U	D	S	A
Sensor														
Physical Jamming/Outage			X		X			X		X				
Replay attacks	X	X		X						X				
Data drops			X	X				X	X					
Spoofing	X	X					X			X				
Quantization errors	X	X										X		
Actuators														
Jamming			X					X		X				
Stealthy attacks	X	X					X			X				
Malicious data injection	X	X					X			X				

Detection mechanisms	Value			Timing		Presence		Quantity			Subtlety		Replication	
	H	L	U	E	L	C	O	SI	S	Sv	U	D	S	A
Detection mechanisms														
Sample-based	X	X		X	X	X		X	X	X			X	
Hollman-based	X	X			X			X	X				X	
Statistics-based	X	X						X	X		X		X	

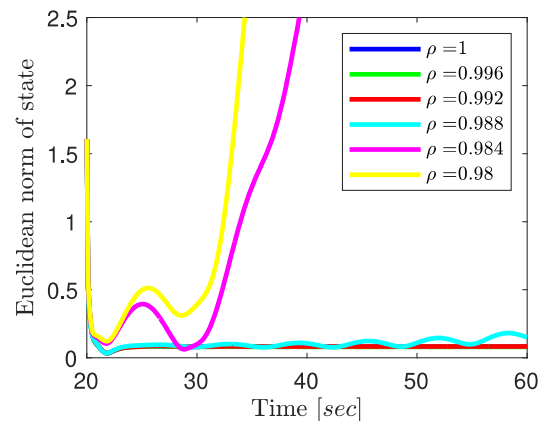
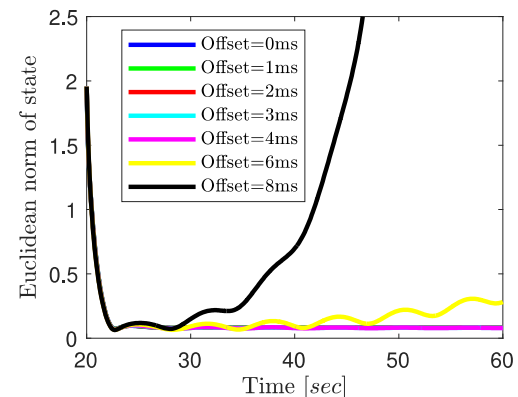
Contribution #2: Impact of Clock Offsets on RL Components

To be published at the [2022 American Control Conference](#), with GeorgiaTech

Research question: A CPS gathers rely on multiple sensors, interconnected through buses and CPUs. The architecture may induce delays and jitters that are clock offsets. Could these impact Reinforcement Learning (RL) based controllers?

Contributions:

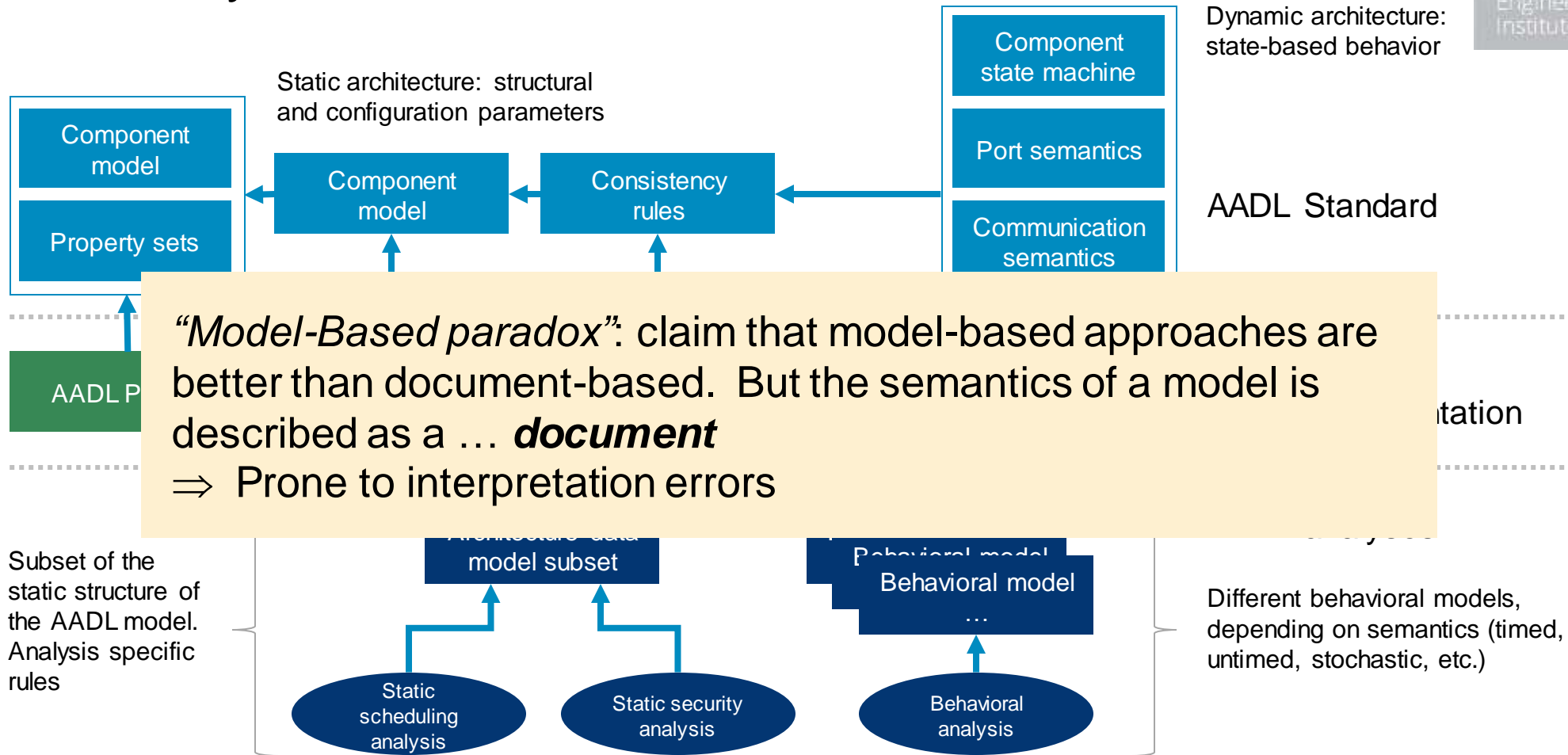
- If clock offsets are bounded, RL can still converge, with theoretical proof without quantification, simulation results show limits in offsets that ensure convergence
- Quantization errors also impact RL convergence



LSI SAFIR Contributions

1. Fault Detection Isolation and Recovery provides the foundation to safety
- 2. Mechanized semantics of architectural description**
⇒ **Coq mechanization of AADL semantics: static, behavior, time, error**
3. Generating arguments about system safety

AADL Layers



AADL Mechanization in Coq

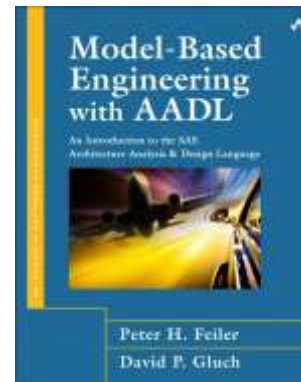
Research question: Provide unambiguous formal semantics for AADL

- Reference for other tools
- Improved standard by eliminating corner cases

Solution: Mechanize the semantics of AADL using the Coq Interactive Theorem Prover (ITP)

- Static and dynamic semantics, property sets

Oqarina released as software artefact:
github.com/Oqarina under the BSD (SEI) license.

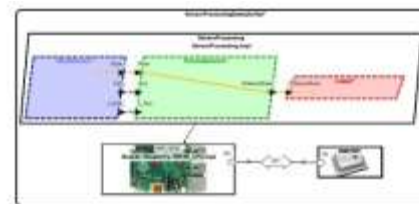


```

Inductive component :=
| Component : identifier -> (* classifier *)
  ComponentCategory -> (* category *)
  identifier -> (* classifier *)
  list feature -> (* features *)
  list component -> (* subcomponents *)
  list property_value -> (* properties *)
  list connection ->
  component

with feature :=
| Feature : identifier -> (* its unique identifier *)
  DirectionType ->
  FeatureCategory -> (* *)
  component -> (* corresponding component instance *)
  list property_value -> (* properties *)
  feature

with connection :=
| Connection : identifier ->
  list identifier -> (* path to the source feature *)
  list identifier -> (* path to the destination feature *)
  connection
  
```



SAFIR delivers formal semantics of AADL as Coq types, theorems, and operational semantics.

Oqarina

Import model from
AADL toolsets

AADL meta-model
as Coq types

Is the model “correct” with
regards to static semantics
rules (e.g., types, static
composition)

Behavior of
component
categories, dynamic
composition rules



Coq data types

⇔ AADL meta-model, typing rules, support for building a model

Well-formedness rules

⇔ AADL legality/consistency rules (i.e., model validity)

Operational semantics

⇔ how to “execute” a model (e.g., proof, model checking, simulation)

Features:

- Simulation of an AADL model by mapping to the DEVS formalism
- Mono-core scheduling analysis using the PROSA library
- To come: fault propagation and analysis

LSI SAFIR Contributions

1. Fault Detection Isolation and Recovery provides the foundation to safety
2. Mechanized semantics of architectural description
- 3. Generating arguments about system safety**
 - ⇒ **Extend ALISA to generate Goal Structuring Notation reports**

Argumentation

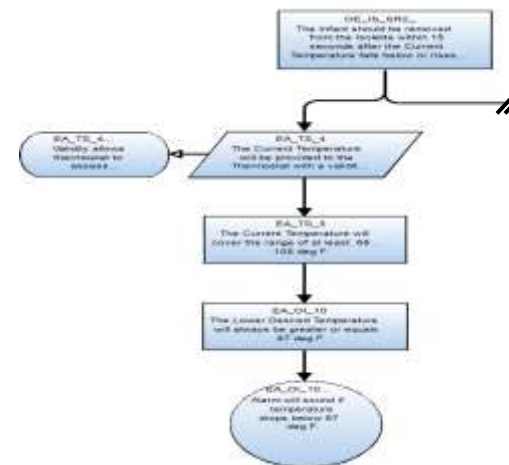
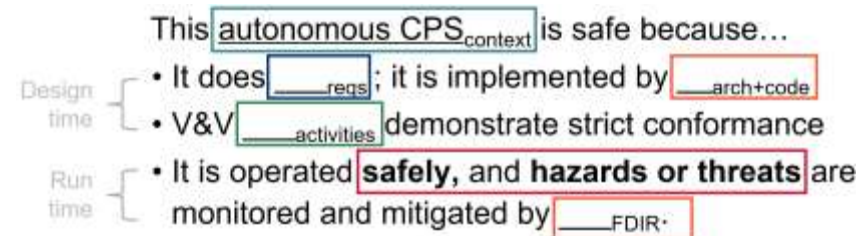
Safety argumentation requires a clear and unambiguous representation, beyond plain English

Main Objective:

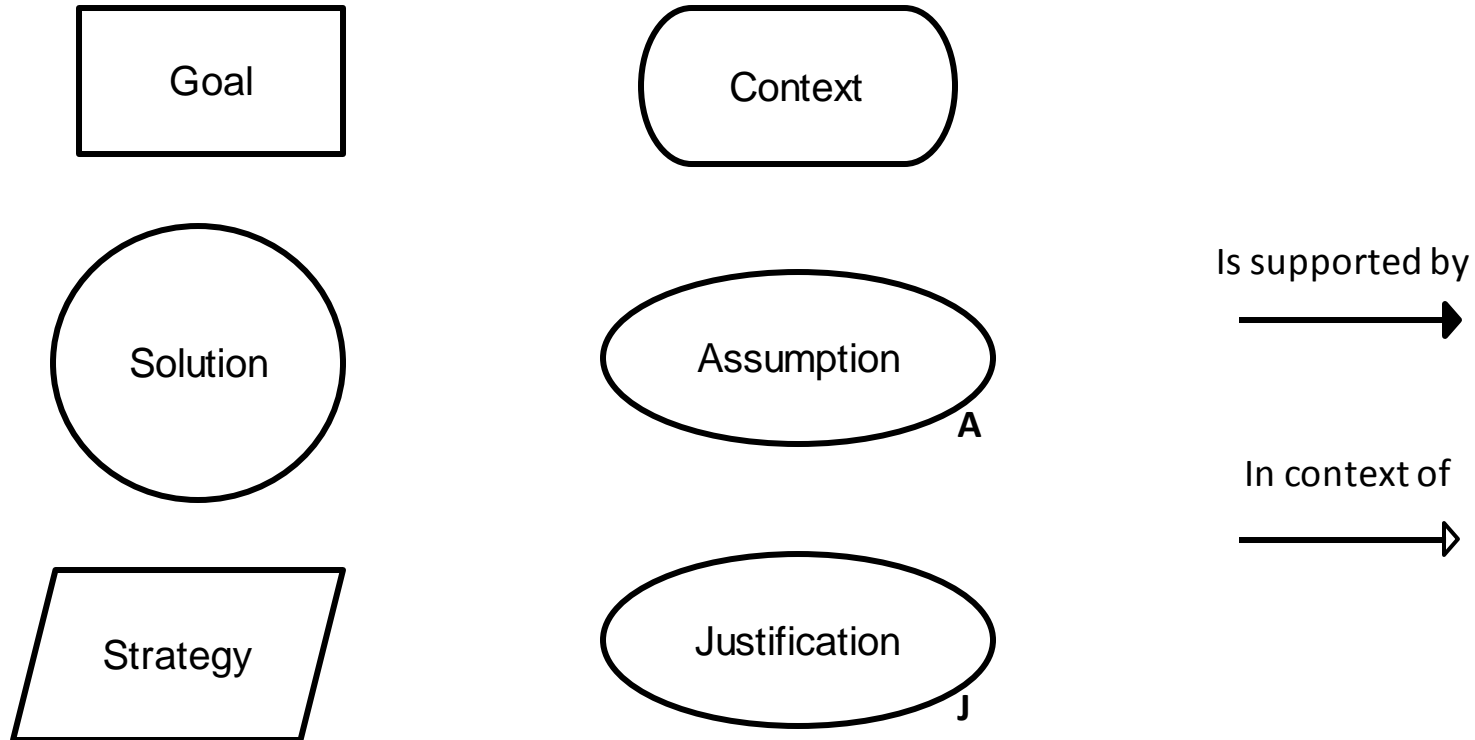
- Increase readability and understanding of safety arguments

Gives developers freedom to use:

- AADL modeling language
- Supported OSATE analyses
- ALISA's rigorous assurance verification
- Generated clear and concise GSN arguments



Goal Structuring Notation



Tim Kelly and Rob Weaver. The Goal Structuring Notation – A Safety Argument Notation
 Proc. of Dependable Systems and Networks 2004 Workshop on Assurance Cases
<https://www-users.cs.york.ac.uk/tpk/dsn2004.pdf>

ReqSpec SEI TR

```

val LowerAlarmBound = 93.0
val LowerDesiredBound = 97.0
val UpperAlarmBound = 103.0
val UpperDesiredBound = 100.0

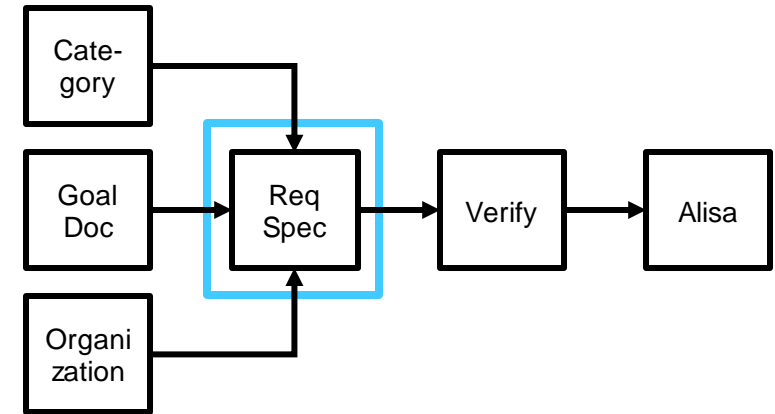
requirement OE_IS_SR2_for alarmResponse [
  val NurseAlarmResponse = 15 sec
  compute ActualLatency: Time
  compute SpecLatency : Time
  compute Mode : string
  description "The infant should be removed from the Isolette within 15 seconds after the Current Temperature falls below or rises above the Alarm Temperature Range."
  category GSN.Goal
  see goal TempGoals.GSN1
  value predicate ActualLatency <= NurseAlarmResponse
]

requirement EA_HS_1 for heat_controll
description "The heat source has a control variable Heat Control which can be set on/off."
rationale "Prolonged exposure of Infant to unsafe heat or cold. Classification: catastrophic."
// validation: presence of incoming feature and data type supports on/off
category GSN.Goal
refines OE_IS_SR2_for
see goal TempGoals.GSN1
]

```

Refers to a category file

Connects EA_HS_1 to top level requirement



Manages stakeholder and system requirements

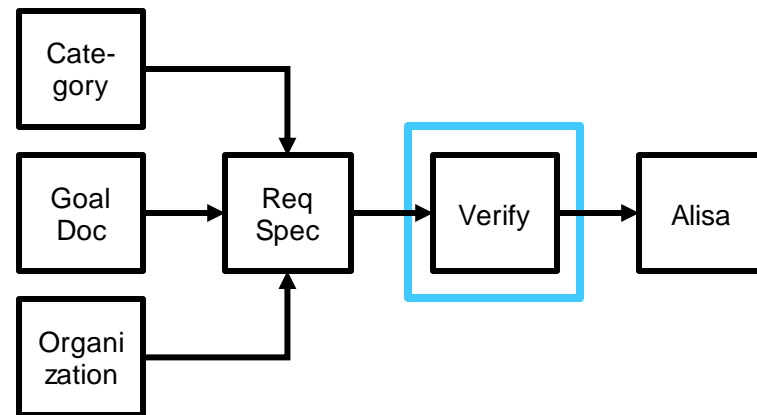
Verify

```

claim EA_HS_1 [
  rationale "Heat source ability to turn on and off is valid"
  activities
    isOnOffHC: ResoluteIsolette.IsOnOff()
]

```

- Verification of requirement EA_HS_1
- **Rationale** keyword is used as a description of verification activities
- **Activities** keyword calls verification methods
 - Verification methods are call analyses built into OSATE to verify requirements against the AADL model



ALISA to GSN Mapping

ReqSpec: System requirements

Description: *A textual description of the req.*

Category: *List of category references*

Refines: *References to other reqs. that this req. refines*

Rationale: *The rationale for the requirement*

Verify: Claims to show reqs. are met

Rationale: *The rationale for the claim*

GSN: Assurance case

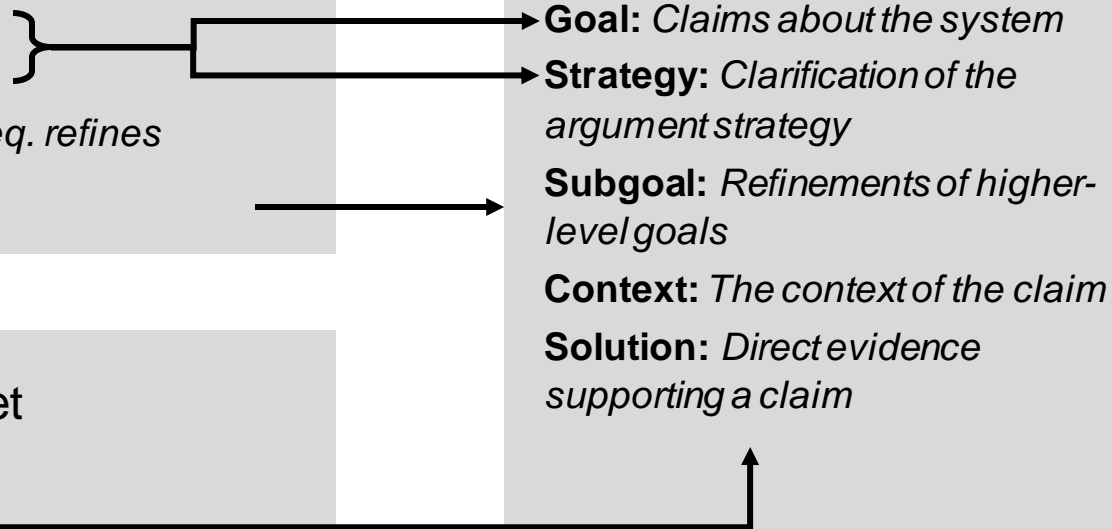
Goal: *Claims about the system*

Strategy: *Clarification of the argument strategy*

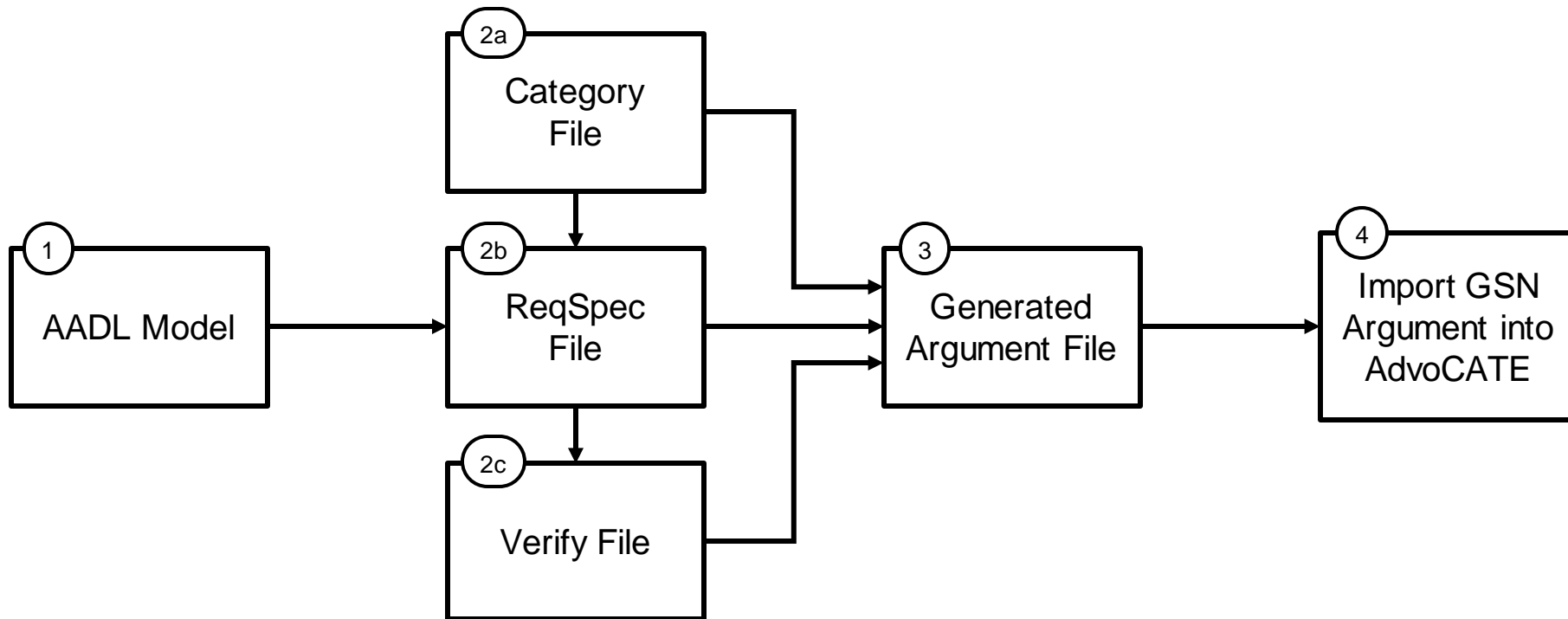
Subgoal: *Refinements of higher-level goals*

Context: *The context of the claim*

Solution: *Direct evidence supporting a claim*



Workflow



SAFIR contributions



SAFIR extends virtual integration capabilities of MBSE (a.k.a shifting to the left) for AI CPS

#1: FDIR with and for AI function **[Reports]**

#2: Semantics of CPS Architecture improved V&V, simulation **[Software]** Oqarina, mechanization of AADL in Coq

#3: Reports generation part of ALISA assurance process **[Software]** OSATE output arguments in the GSN format

Our Team



Jerome Hugues
Principal Investigator,
Sr. Architecture Researcher



Aaron Greenhouse
Sr. Architecture Researcher



Keaton Hanna
Assistant Software Engineer



Sam Procter
Sr. Architecture Researcher



Joseph Seibel
Member of the Technical Staff



Lutz Wrage
Senior Member of the
Technical Staff

Contact Us

To work with us, contact
info@sei.cmu.edu

Document Markings



Copyright 2022 Carnegie Mellon University.

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.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

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.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and 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.

DM22-0882