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

Workshop #1 – May 2022

PI: 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.

DM22-0376

Objectives: collecting feedback on LSI SAFIR

Id Card of an SEI LINE Project

- SEI-funded research under both 6.2/6.3
- Duration: 3 years
- Level of effort: 4 ETP/year
- Expected impact: TRL 5, derived from AFRL definitions
  - TRL 5P: There is clearly a significant, specific DoD need that can be met by the technology
  - TRL 5A: The project's technology is sufficiently efficient for use in surrogate DoD scenarios at scale

The Safety-Critical Embedded Software System Challenge

Problem:

- Software increasingly dominates safety and mission-critical system development
- Issues discovered long after they are created

Goal:

Early discovery of system-level issues through virtual integration and incremental analytical assurance

Solution:

- **Language** standardized via SAE International & matured into practice through pilot projects & industry initiatives
- **Tooling** available under open source license continually enhances analysis, verification, and generation capabilities
- **Expertise** in Modeling Safety-Critical Embedded Systems

Model-Based Engineering for Cyber-Physical Systems

Create the best design that holds up over time as the system evolves. + Test the design without having to write any code. = Build a single model to assess hardware and embedded software before the system is built.

SAE AADL / ACVIP

- Standardized language and process for the engineering safety-critical systems.

OSATE

- Open Source AADL toolset for performing verification and validation (V&V).

DoD Transitioning

- Maturity increased through pilot projects and trainings.

Outline

1. Introduction of the SAFIR project
2. Safety considerations for AI-enabled systems
3. Oqarina: mechanization of the AADL language using the Coq theorem prover
4. ASAP: the Architecture-Supported Audit Processor
5. Wrap-up discussion

The IA/AI fresh breeze .. coming from an iceberg ..

General enthusiasm on Increasingly-Autonomous CPS [1] to

- Improve system efficiency (decrease # operators), system capability (automate high-level tasks), faster than human, ..

Increa
"intelli

But fa
makes

SAFIR aims to
a) incorporate advanced **S**afety **A**nalysis techniques to
b) synthesize **F**ault Detection, **I**solation and **R**ecovery policies for time-sensitive IA-CPS

⇒ 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.

On safety and security of Increasingly Autonomous CPS

Safety of an autonomous system can be decomposed into security, safety, and correctness arguments (definitions from [IEC])

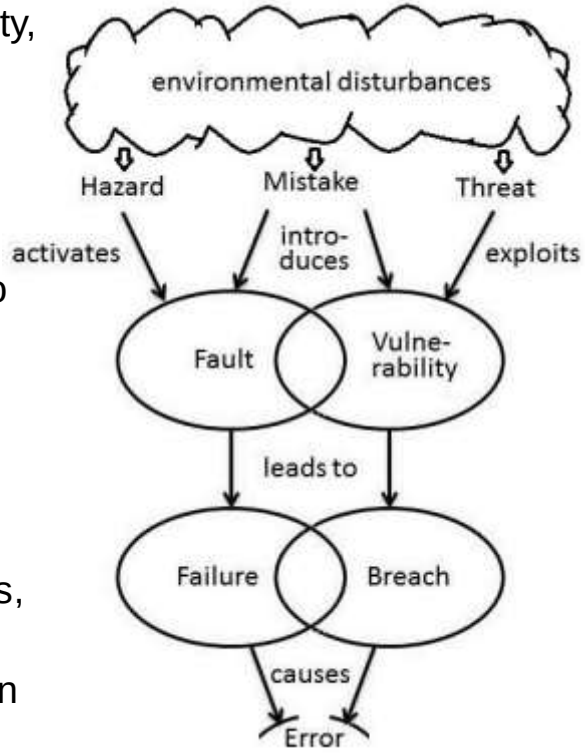
- Safety is concerned when a failure leads to severe consequences (high risk) for the operational environment
- Security is concerned when a failure to protect assets (a breach) leads to severe consequences for the system itself (and potentially to its operational environment).
- Correctness is concerned with the realization of a function

SAFIR focus on the engineering of safety-critical IA-CPS

Ensure Safety and security properties at the architectural-level, i.e.,

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

ensure the architecture properly mitigates faults, preserving the mission

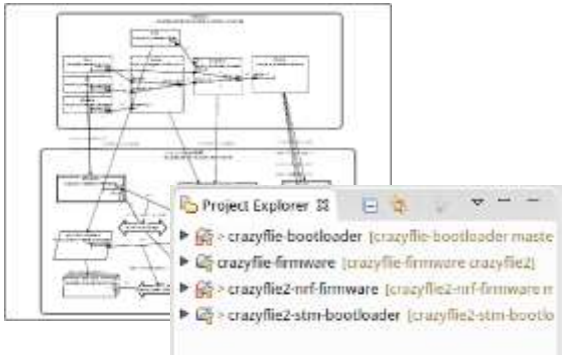


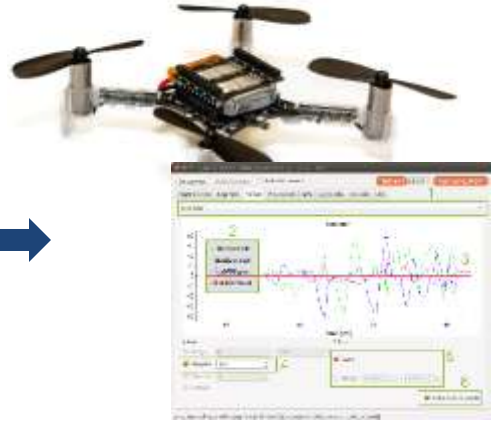
From doi:10.3390/info7020033

LSI SAFIR ”Big Picture”

Safety of Increasingly Autonomous Systems

+





“From a safety standpoint, a system architecture embodies the principles and design decisions by which system, sub-system, item, and their emergent behavior can be reliably constrained to an operational state space that is acceptably safe”. SAE AS6983

⇒ SAFIR does not consider the V&V of LEC components, but the V&V of their integration within a general CPS architecture

LSI SAFIR "Big Picture" Work Breakdown structure

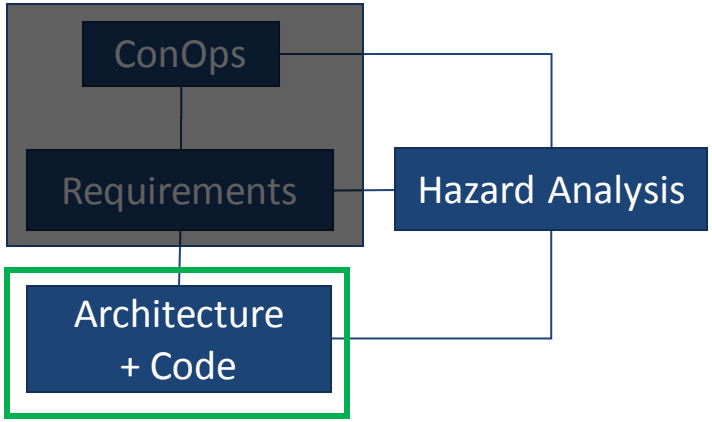
Safety assurance is a multi-tier, multi-domain endeavor

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.

Existing SoA



LSI SAFIR is building a comprehensive approach to
- Support both Systems Engineering and Safety Assessment processes *through*
  - tool-support, architectural patterns at both model and runtime levels, new analysis capabilities, *and*
  - an argument the above are self-consistent

LSI SAFIR Contributions

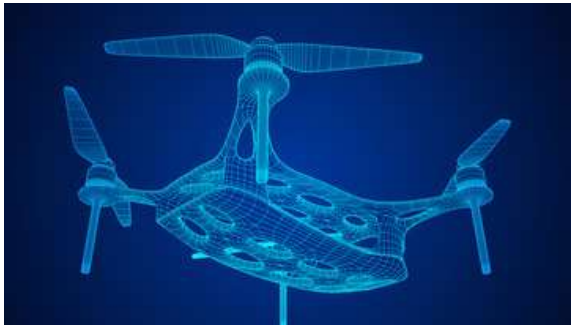
1. Fault Detection Isolation and Recovery provides the foundation to safety
   ⇒ Fault/attack detection mechanisms based on Reinforcement-Learning
   ⇒ FDIR patterns in the scope of autonomous systems
   ⇒ Reference architecture for FDIR-capable systems

2. Safety assurance relies on properties asserted on a system architecture
   ⇒ Coq mechanization of AADL semantics: static, behavior, time, error

3. Analyzing system safety
   ⇒ Views (slices) of a system to review hazards and their impact
   ⇒ Derived from / align with STPA, can be extended to support more viewpoints

LSI SAFIR Case Study – 6.2 variant

10.1109/ACCESS.2019.2909530



Let us consider a patrolling mission with the following:

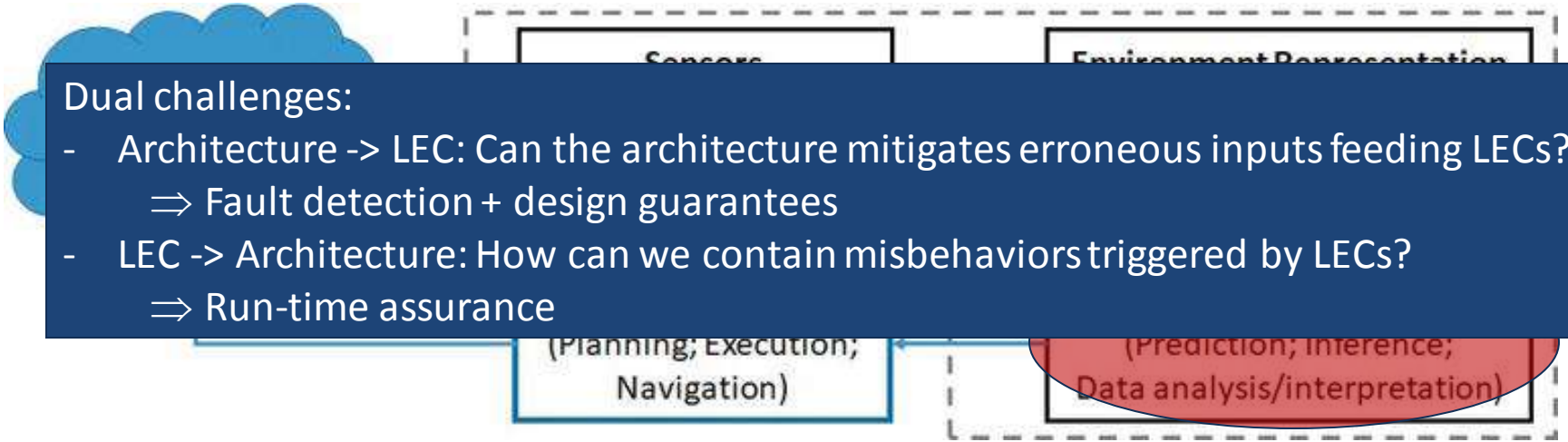
- Partially known place: a factory, with slow-moving parts
- Both closed and open areas: wind, lighting conditions
- Find and detect intruders, recognize threats
- Tight maneuvers to enter/exit buildings, safety margins to avoid damages
- Autonomy in decision making, basic man-machine teaming
  1. classify threats, notification of threats to operator
  2. select next step: navigation plan / follow threat / back home for maintenance
  3. autonomous flight with switch to either human operated or fail-safe mode

FY21-22: theoretical foundations to support this case study

FY23: realization in a controlled environment with Georgia Tech

Challenges in LEC integration: Dual perspective

Timing? Value?



Dual challenges:

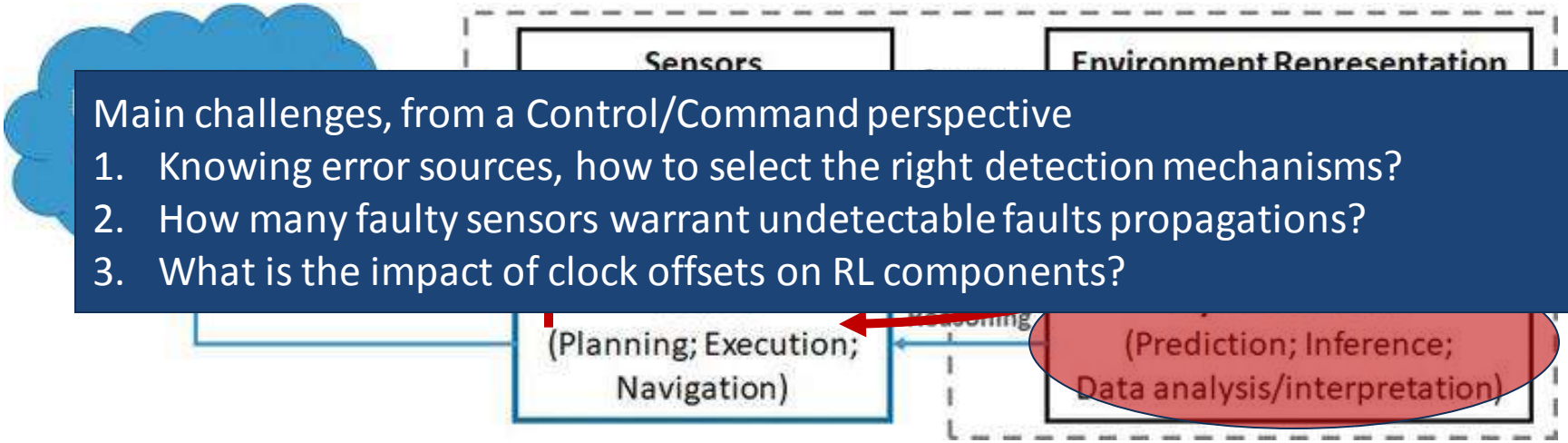
- Architecture -> LEC: Can the architecture mitigates erroneous inputs feeding LECs?
  ⇒ Fault detection + design guarantees
- LEC -> Architecture: How can we contain misbehaviors triggered by LECs?
  ⇒ Run-time assurance

Accuracy/confidence?
WCET
Sensitivity to data

Adapted from DOI: 10.5772/intechopen.79742

Challenges in LEC integration: Control/Command perspective

Timing? Value?



Main challenges, from a Control/Command perspective

1. Knowing error sources, how to select the right detection mechanisms?
2. How many faulty sensors warrant undetectable faults propagations?
3. What is the impact of clock offsets on RL components?

Accuracy/confidence?
WCET
Sensitivity to data

Adapted from DOI: 10.5772/intechopen.79742

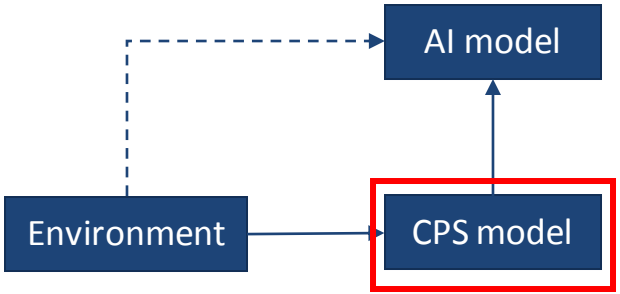
Fault Taxonomy Classification of misbehaviors in LEC-CPS

Notionally, the AI function receives data from the environment, take a decision or act on it

Pragmatically, this data is process 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
- And pre-existing faults in non-AI CPS

[1] provides a research agenda, testing existing patterns for fault tolerance but recognize the need for specialization for ML, especially in fault characterization and detection



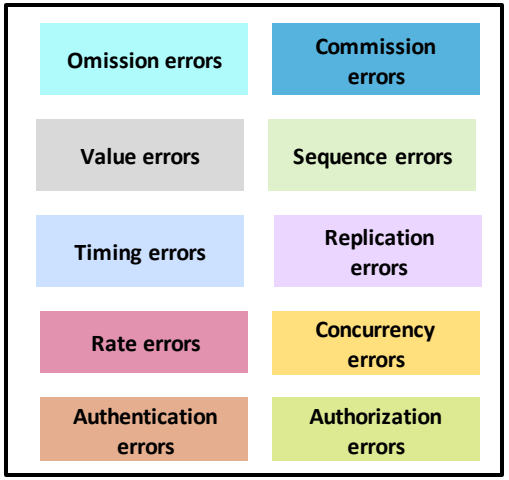
[1] Myllyaho, Lalli, Mikko Raatikainen, Tomi Männistö, Jukka K. Nurminen, and Tommi Mikkonen.
"On Misbehaviour and Fault Tolerance in Machine Learning Systems."
Journal of Systems and Software 183 (January 2022): 111096.
https://doi.org/10.1016/j.jss.2021.111096.

Challenge#1: fault/attack detection

Faults (safety) and attacks (security) in LEC-CPS: different causes/same impact

Can be characterized using AADL EMV2 fault taxonomy

Fault Propagation Taxonomy
Part of SAE AADL Standard Suite



Challenge#1: fault/attack detection

Published in https://doi.org/10.2514/6.2022-0969, with GeorgiaTech

EMV2 fault taxonomy as pivot between faults/attacks and detection mechanism

- Scenario defines a set of error it produces: error_set {scenario}
- Detection mechanism a set of error it mitigates error_set {detector}

Condition:

- error_set {detector}.⊃ error_set {scenario}

⇒ Contribution: Survey of fault detectors, mapping of fault/attack scenario, to be extended for actual case studies or systems.

Faults	Value			Timing		Presence		Quantity			Subtlety		Replic
	H	L	U	E	L	C	O	Sl	S	Sv	U	D	S
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		Replic
	H	L	U	E	L	C	O	Sl	S	Sv	U	D	S
Detection mechanisms													
Sample-based	X	X		X	X	X		X	X	X		X	
Bellman-based	X	X			X				X	X		X	
Statistics-based	X	X								X		X	

Challenge #2: conditions for fault/attack detectability

To be published at the 2022 American Control Conference, with GeorgiaTech

Security index quantifies the *least* effort to conduct perfectly undetectable attacks.

Consider a linear time invariant system with attack vector a_k

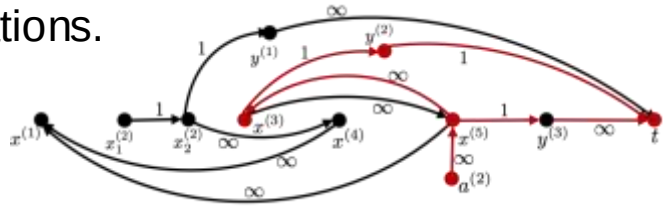
$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k + B_a a_k, \\ y_k &= Cx_k + D_a a_k, \end{aligned}$$

For the system $\Sigma = (A, B, C, B_a, Da)$, the security index is defined as the minimal number of attacked sensors and actuators to conduct perfectly undetectable attacks,

$$\begin{aligned} s_0 = \min_{a_k} &||a_k||_0 \\ \text{s.t.} \quad x_{k+1} &= Ax_k + B_a a_k, \\ \mathbf{0} &= Cx_k + D_a a_k, \\ x_0 &= \mathbf{0}, \end{aligned}$$

We show that computing the security index is equivalent to a max-flow problem on a graph built from the system equations.

s = 2 ⇔ 2 actuators must be compromised to forge an undetectable attack or fault



Challenge #3: impact of clock offsets on RL components

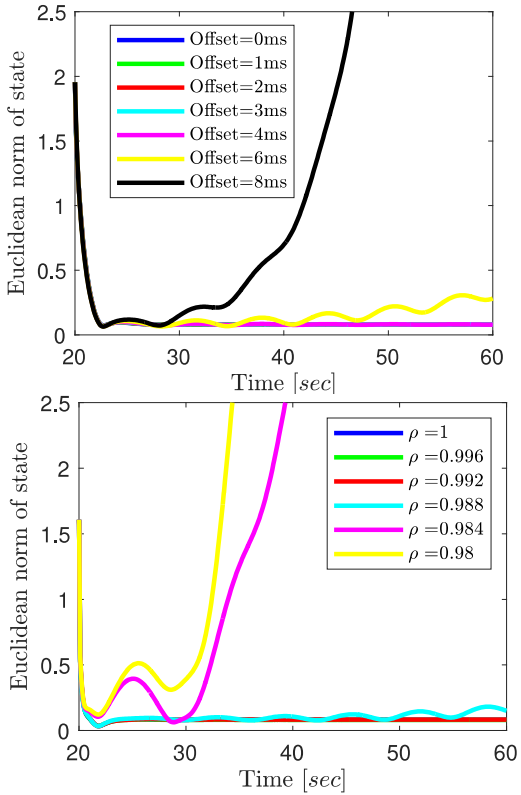
To be published at the 2022 American Control Conference, with GeorgiaTech + another preprint

Research question: a LEC-CPS gathers multiple source of sensors. The architecture may induce delays and jitters that are clock offsets. Could these impact RL-based controllers?

⇒ If clock offsets are bounded, RL can still converge, with theoretical proof without quantification, simulation results show limits in offsets that ensure convergence

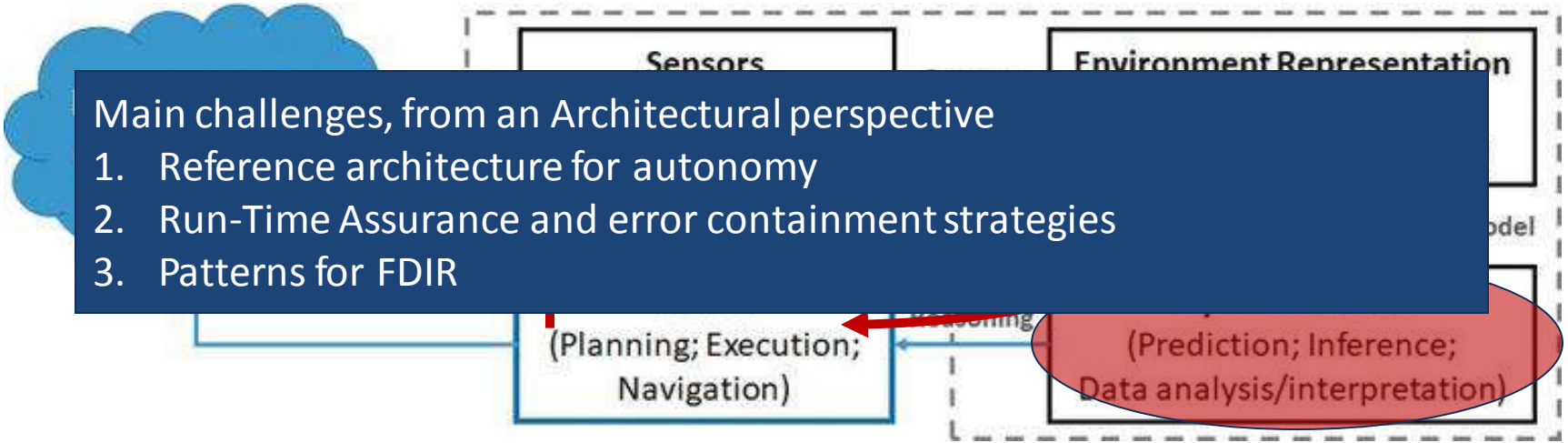
⇒ Quantization errors also impact RL convergence

Open question: upper-bound characterization, otherwise, use simulation



Challenges in LEC integration – Architectural perspective

Timing? Value?



Main challenges, from an Architectural perspective

1. Reference architecture for autonomy
2. Run-Time Assurance and error containment strategies
3. Patterns for FDIR

Accuracy/confidence?
WCET
Sensitivity to data

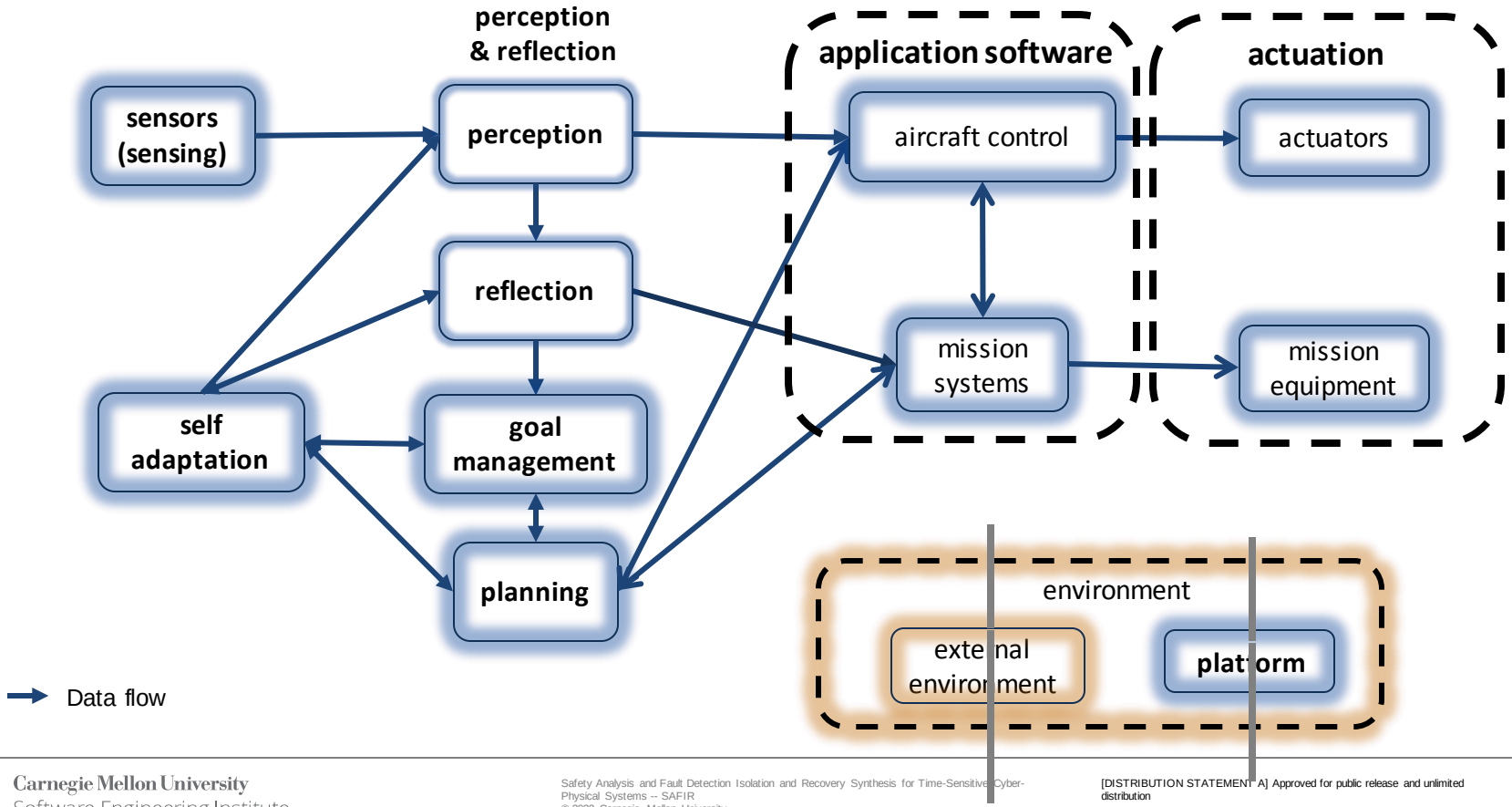
Adapted from DOI: 10.5772/intechopen.79742

Autonomous Systems – An Architectural Characterization

From Sifakis, https://doi.org/10.1007/978-3-030-21485-2_21

Sensing	collection of raw data from the environment
Perception	includes interpretation of stimuli, removing ambiguity/vagueness from complex input data and determining relevant information
Reflection	Involves building/updating a faithful environment run-time model
* Goal Management	choosing among possible goals the most appropriate ones for a given configuration of the environment model
Planning	developing plans to achieve chosen goals
Self-Adaptation	the ability to adjust behavior through learning and reasoning and to change dynamically the goal management and planning processes
Application Software	includes software for aircraft control and mission system operation
Actuation	influence or control conditions in the environment
Platform	encompasses the computing hardware, including support software (e.g., operating system) and the physical components that comprise the autonomous entity

Autonomous Functions - Aircraft System Architecture



Runtime Assurance for LE-CPS

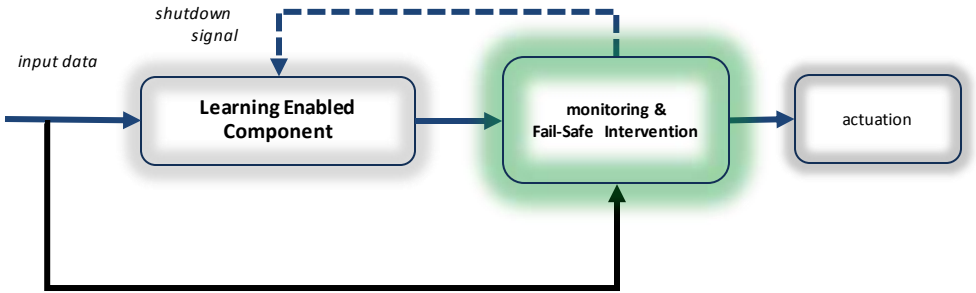
Runtime assurance is a monitoring and intervention approach

Monitoring and intervention capabilities

- employ conventional (non-LEC) technologies.
- are uncomplicated or well established.

These enable verified & trusted implementations

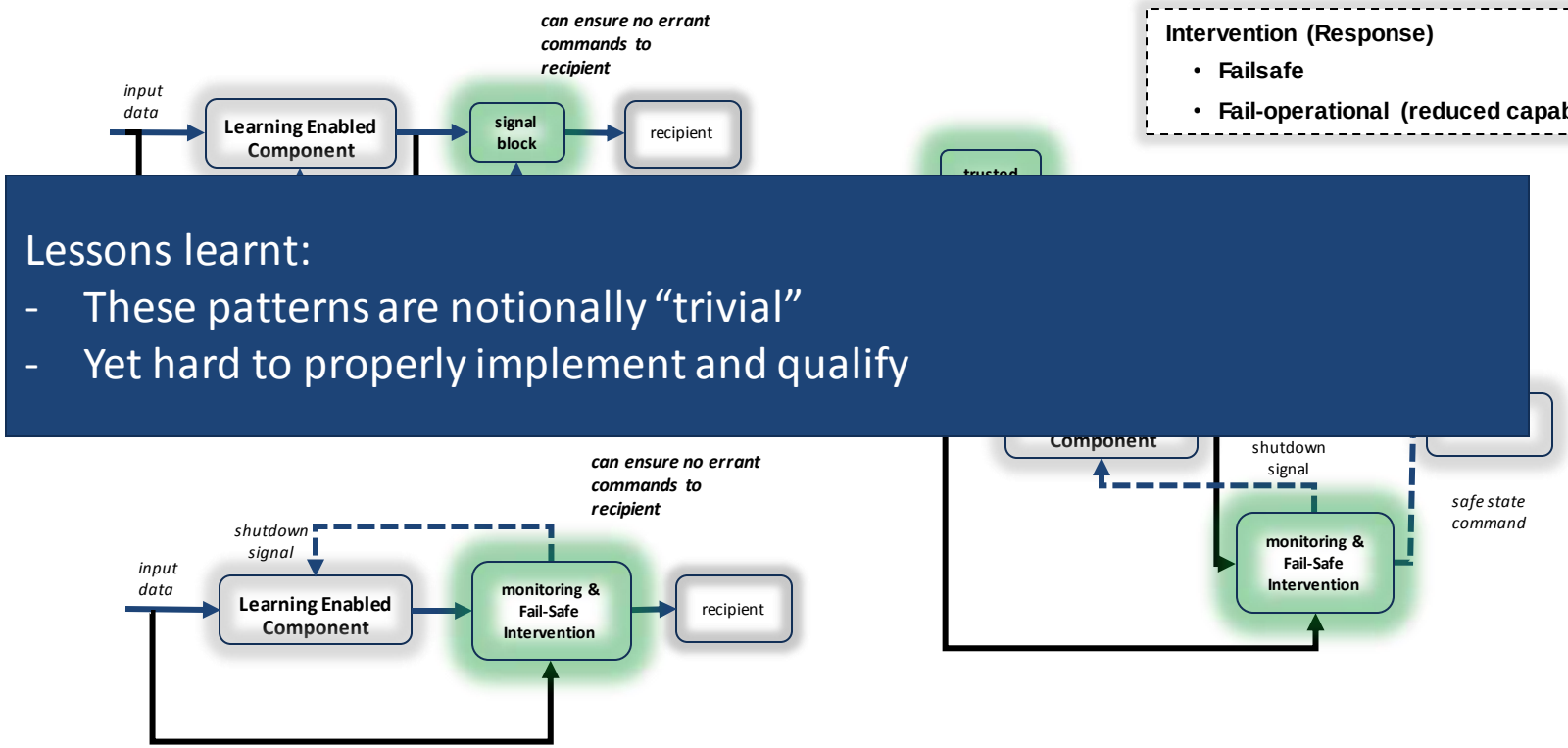
Representative Output Monitoring and Intervention Architecture for Control Applications



trusted

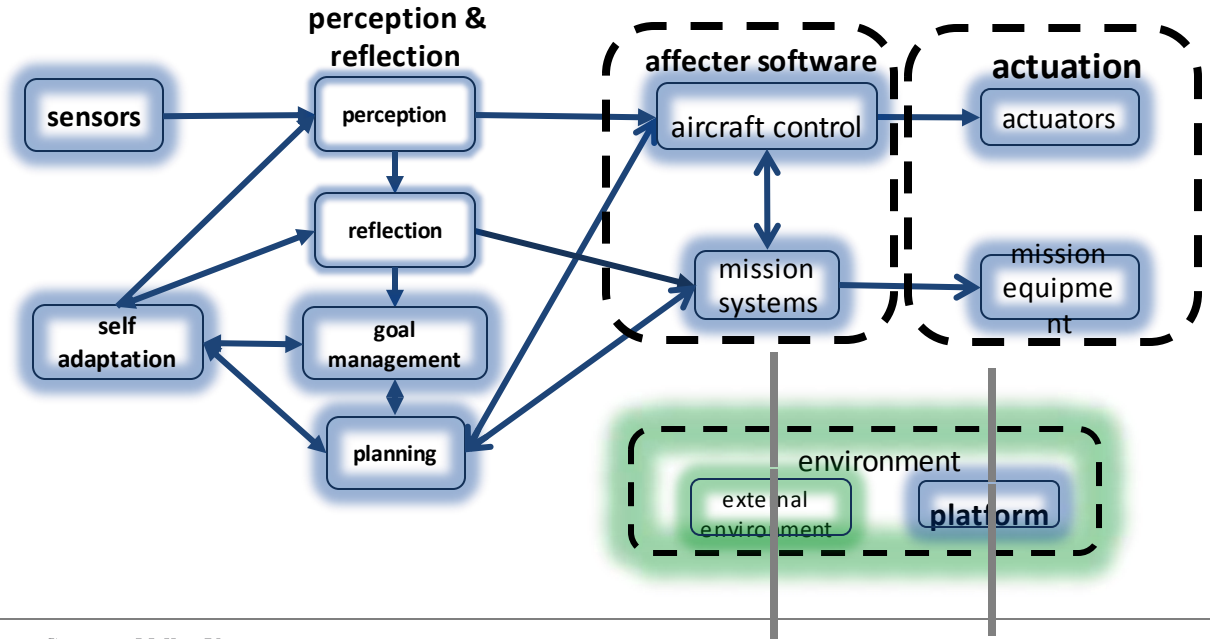
Green Components are 'trusted'
Non-learning enabled technologies

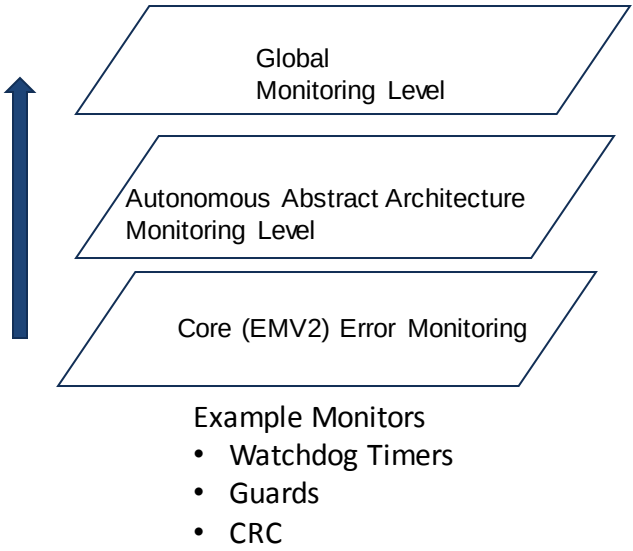
Monitor and Intervention Architecture Patterns – Output Monitoring



FY21-22 Early Results

- Literature survey of reference architecture for runtime assurance;
- Patterns for layering monitoring, assurance, and detection across the architecture





Example Monitors
- Watchdog Timers
- Guards
- CRC

Capturing RTA and reconfiguration patterns -- workplan

Patterns provide a solution to a well-defined architectural concern

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?
- Apply them: how to weave existing architecture with new components

TMR pattern definition

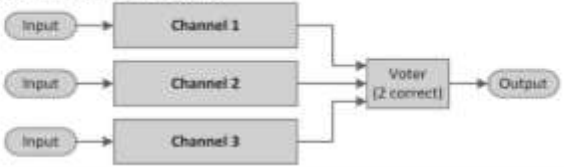
https://doi.org/10.1007/978-3-030-14291-9_2

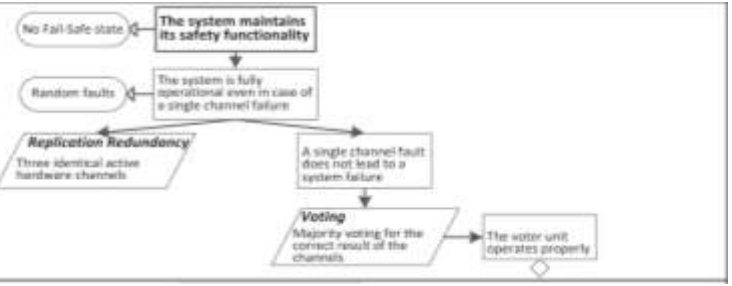
Patterns provide a concise definition to a solution, e.g. Triple Modular Redundancy, Hot/Cold Swap, etc.

Defined in a canonical way,

- Intent, architecture,
- Pro/Cons
- Fragments of a safety argumentation

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.		
Safety GSN			





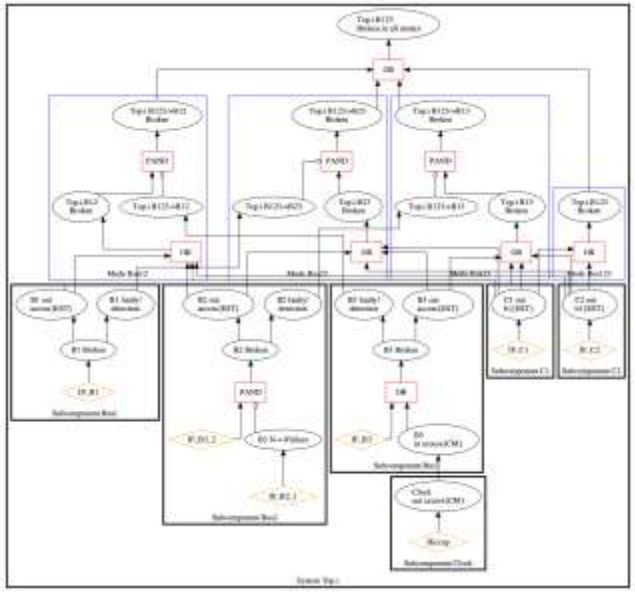
Roadmap FY22-FY23

FY21-FY22: Notional patterns defined and exercised

- Relies on AADL semantics mechanization, see next sect.

FY23:

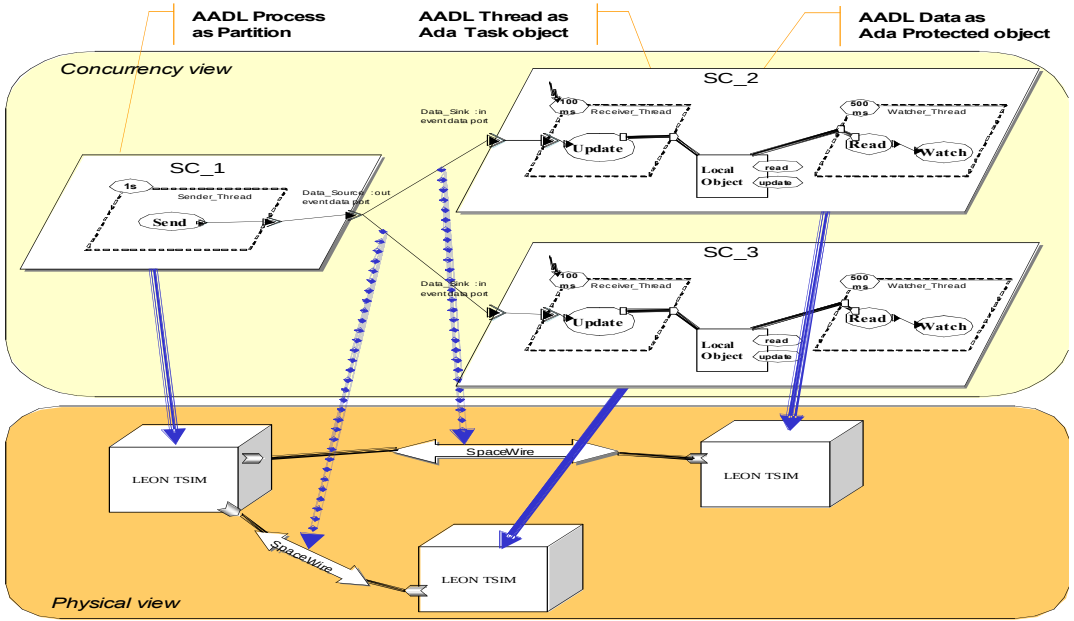
- Full definition of patterns in AADL, covering fault containment and reconfiguration
- Tools extension to support system configuration and reconfiguration (modes)
  - Revisit analyses algorithms, Fault Tree
  - Provide proper GUI for visualization
- Design-space exploration tool to guide designers to apply patterns



Outline

1. Introduction of the SAFIR project
2. Safety considerations for AI-enabled systems
3. Oqarina: mechanization of the AADL language using the Coq theorem prover
4. ASAP: the Architecture-Supported Audit Processor
5. Wrap-up discussion

Why AADL?



General AADL tool: OSATE

Requirements verifications: ALISA

Middleware synthesis : Ada, C (POSIX, ARINC653), RTOS: Ocarina

Scheduling: Gateways to Cheddar, MAST

Error modeling, FTA, FMEA: OSATE

Model checking: Petri Nets (Time and CPN), LNT

Security Analysis, CI/CD, […]

AADL goal is to provide “ground truth” for describing safety-critical systems

SAE International 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*

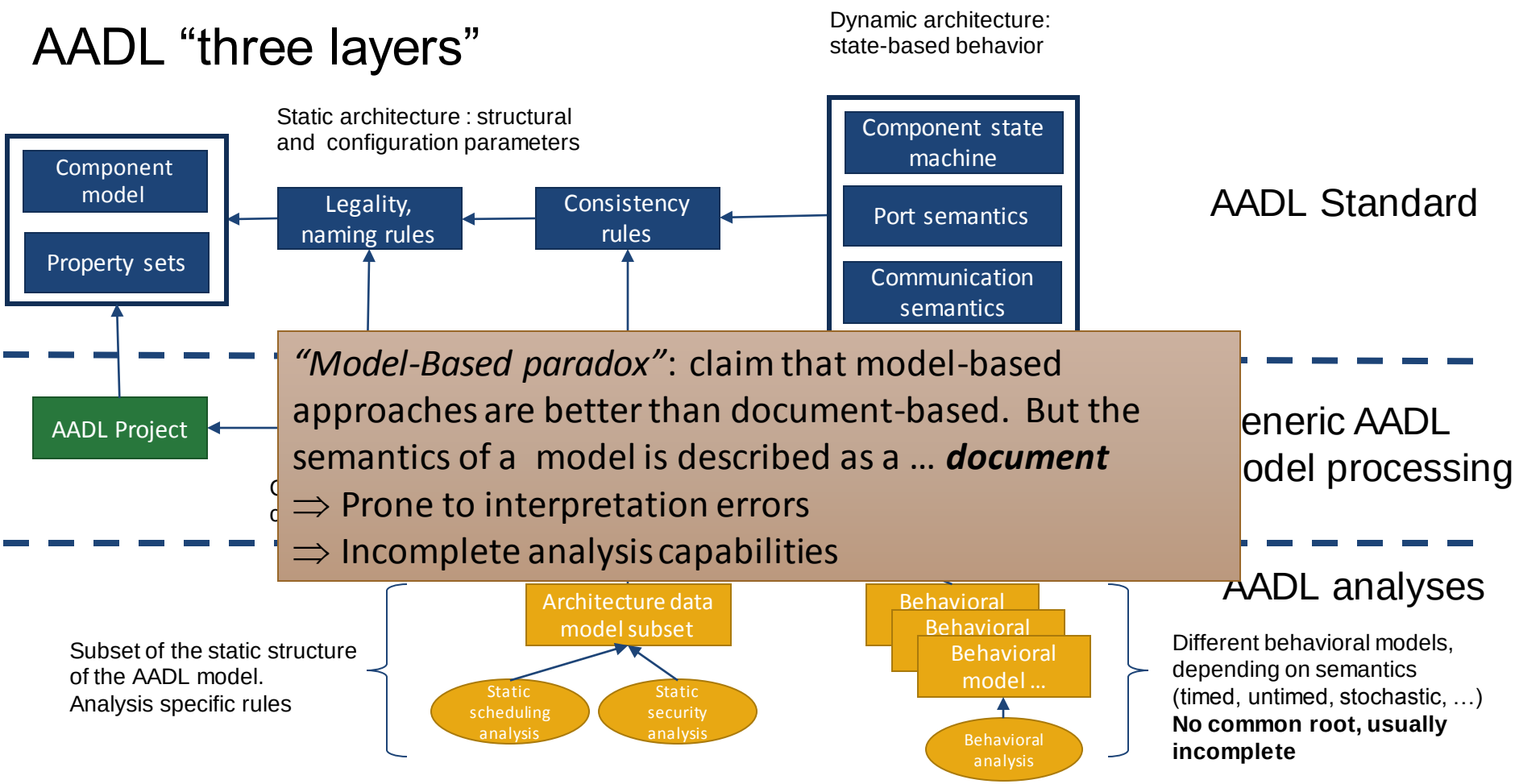
Standardized AADL Annex Extensions
- Error Model language for safety, reliability, security analysis [2006, 2015]
- ARINC653 extension for partitioned architectures [2011, 2015]
- Behavior Specification Language for modes and interaction behavior [2011, 2017]
- Data Modeling extension for interfacing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

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

TQL-5, i.e. verification tools. Output :- evidence, part of some argument

TQL-1 tools, i.e. output is part of final system

AADL "three layers"



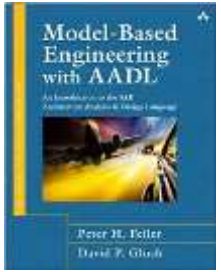
AADL mechanization in Coq

Coq is an interactive theorem prover, well-suited for defining the semantics of programming languages, e.g., the CompCert compiler.

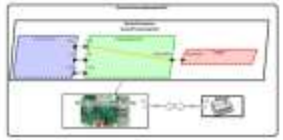
Goal: provide an unambiguous formal semantics for AADL

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

Coq mechanization released as software artefact: https://github.com/Oqarina/ under a BSD license.

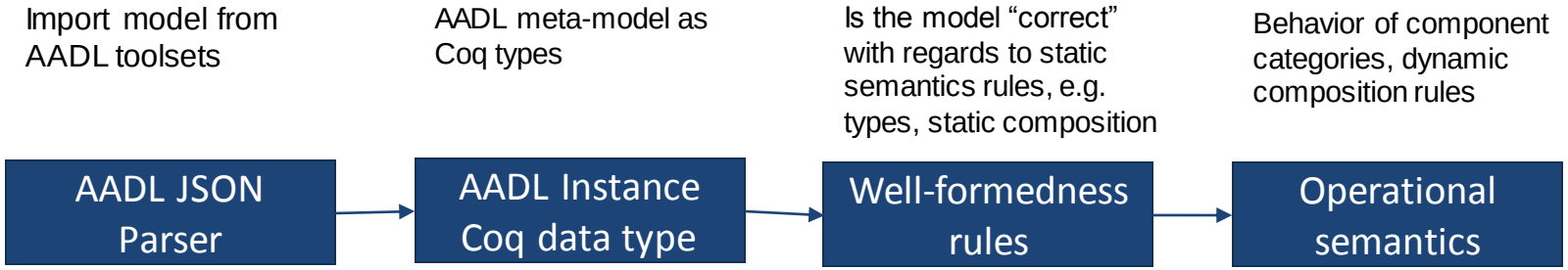



```
Inductive component :=
| Component : identifier → (*  classifier *)
              ComponentCategory → (*  category *)
              identifier → (*  classifier *)
              list feature → (*  features *)
              list component → (*  subcomponents *)
              list property_value → (*  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



Coq data types ⇔ AADL meta-model, basic typing rules, support building a model

Well-formedness rules ⇔ AADL legality/consistency rules, model is valid

Operational semantics ⇔ how to "execute" a model: proof, model checking, simulation, code generation

Coq definitions help improving 1/ AADL standard, 2/ OSATE toolchain.

FY21-22: focus on well-formedness rules and simulation.

FY23: fault analysis, model checking

Summary of AADL mechanization

Problem: safety assurance requires a precise semantics of the system behavior

Solution: mechanize AADL semantics in a theorem prover

Impact: SAFIR delivers formal semantics of AADL as Coq types, theorems, and operational semantics as an Open Source project – Oqarina. AADL practitioners have a reference semantics to build their own tools

FY21-FY22Q2: support core semantics, timing

a. Reasoning/computing on models
b. Simulation using DEVS formalism
c. Code extraction to build standalone tool or direct use of Coq
d. Publication of AADLv2.3

FY22Q2-FY22Q4: addition of error concepts

a. Extension of reasoning and simulation capabilities

Outline

1. Introduction of the SAFIR project
2. Safety considerations for AI-enabled systems
3. Oqarina: mechanization of the AADL language using the Coq theorem prover
4. ASAP: the Architecture-Supported Audit Processor
5. Wrap-up discussion

Objective / Contributions / BLUF

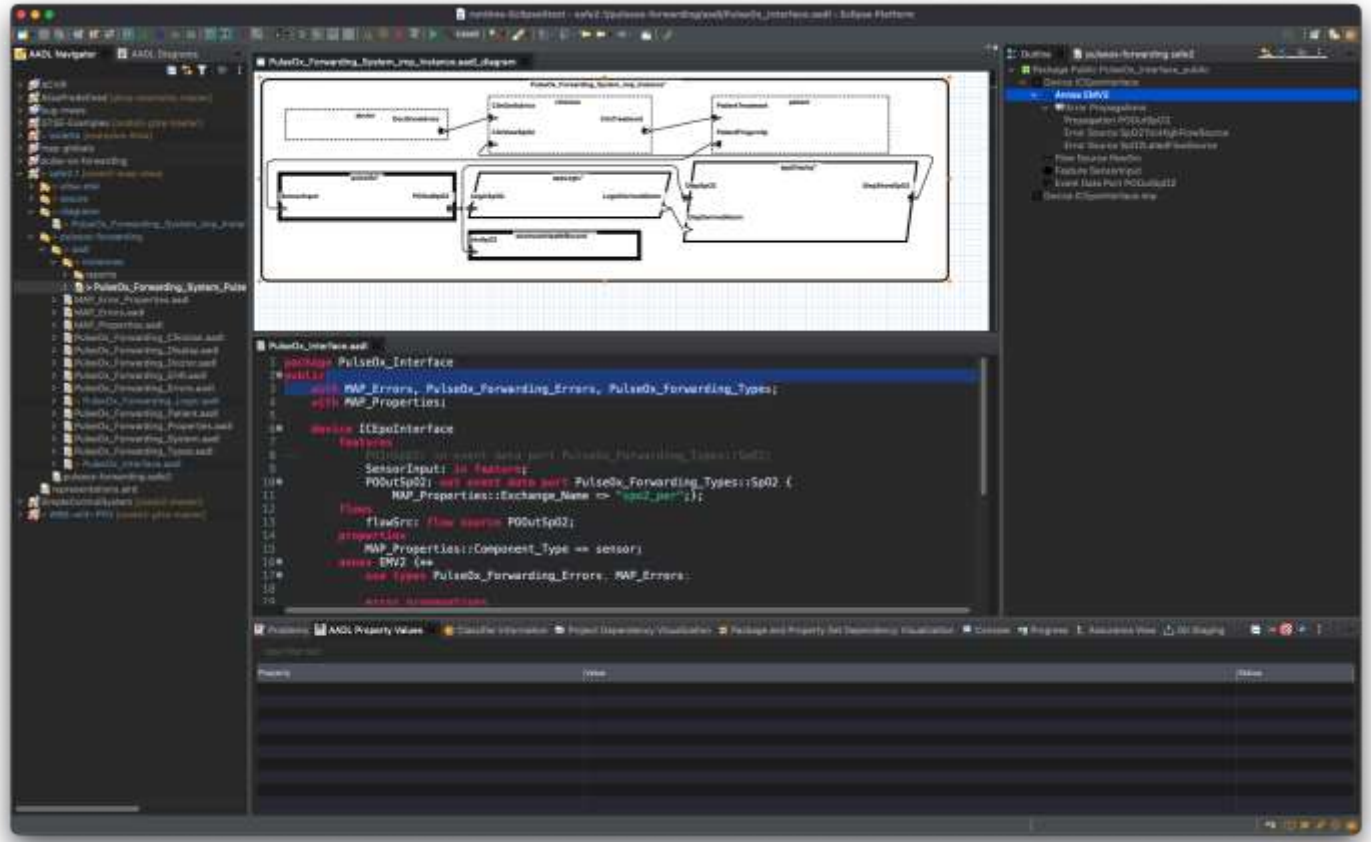
Summary: ASAP is a collection of views of a system. The views are designed to be used for analyzing a system's safety.

- Views are implemented in OSATE, ie, we're discussing tooling that is publicly available now
- Current views are derived from / align with STPA, but more can be added to align with other hazard analysis techniques or standards-based approaches

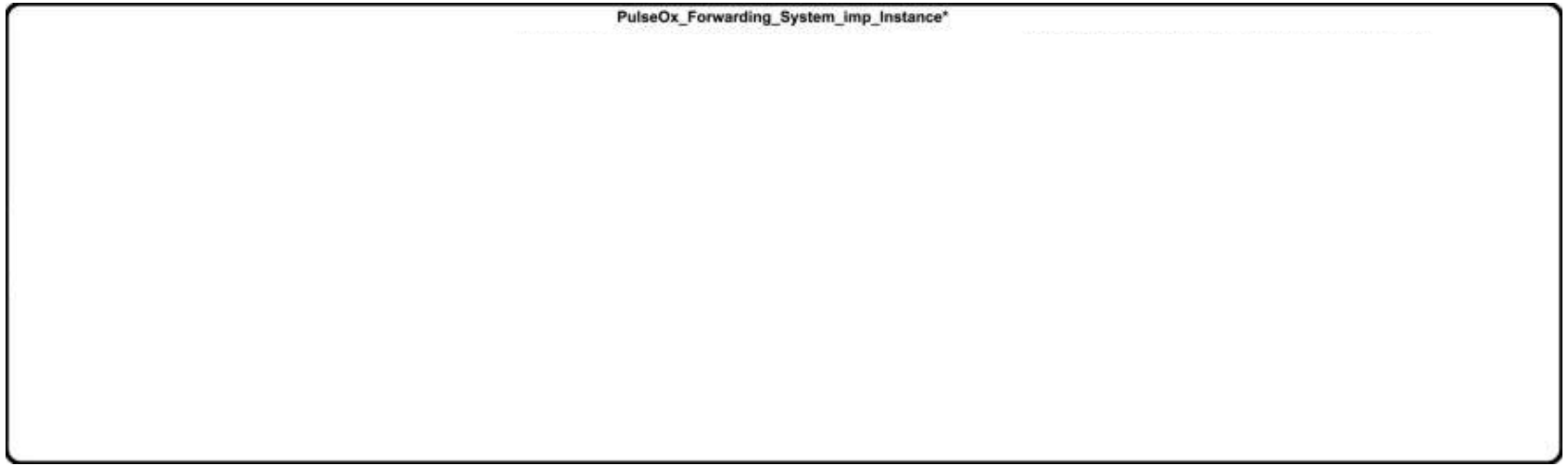
Outline

1. **Background**
    1. AADL / OSATE
    2. PulseOx Forwarding
    3. STPA, SAFE
2. ASAP: Three Viewpoints
3. Future Work

AADL & OSATE

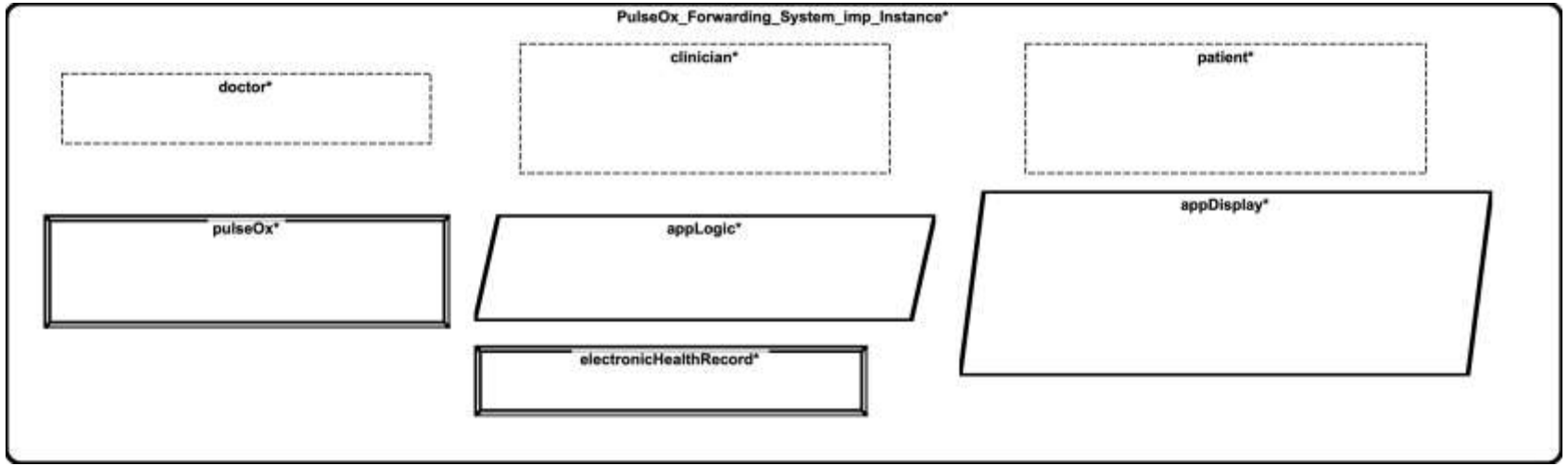


The PulseOx Forwarding Example

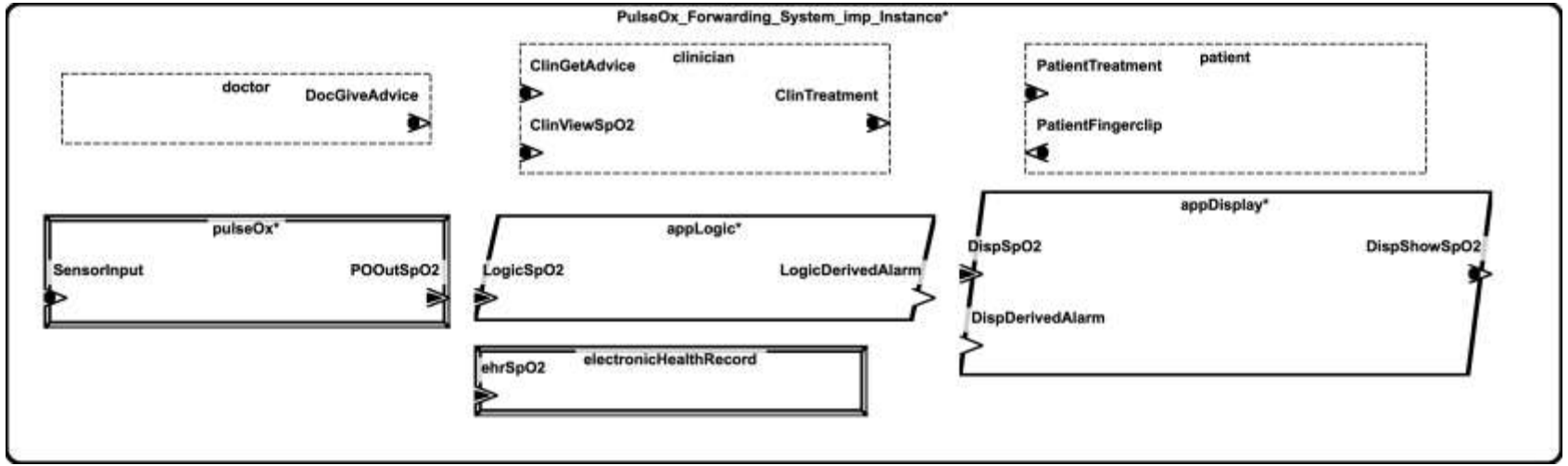


Pulse oximeter reads blood-oxygen saturation from a patient, monitoring software displays an alarm if values are out of expected range

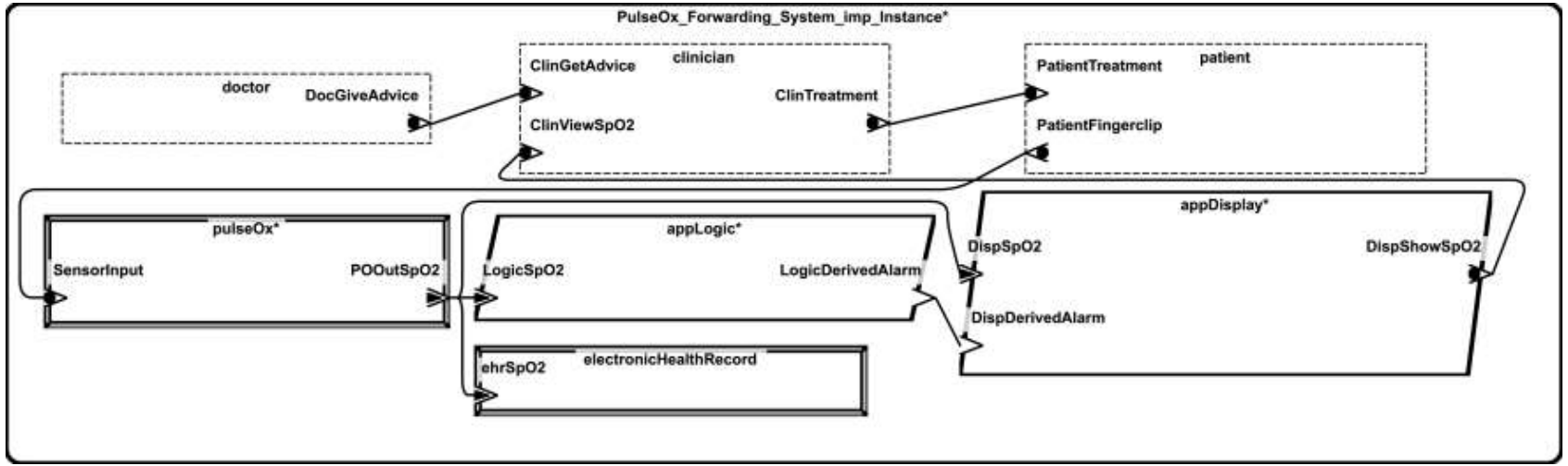
The PulseOx Forwarding Example



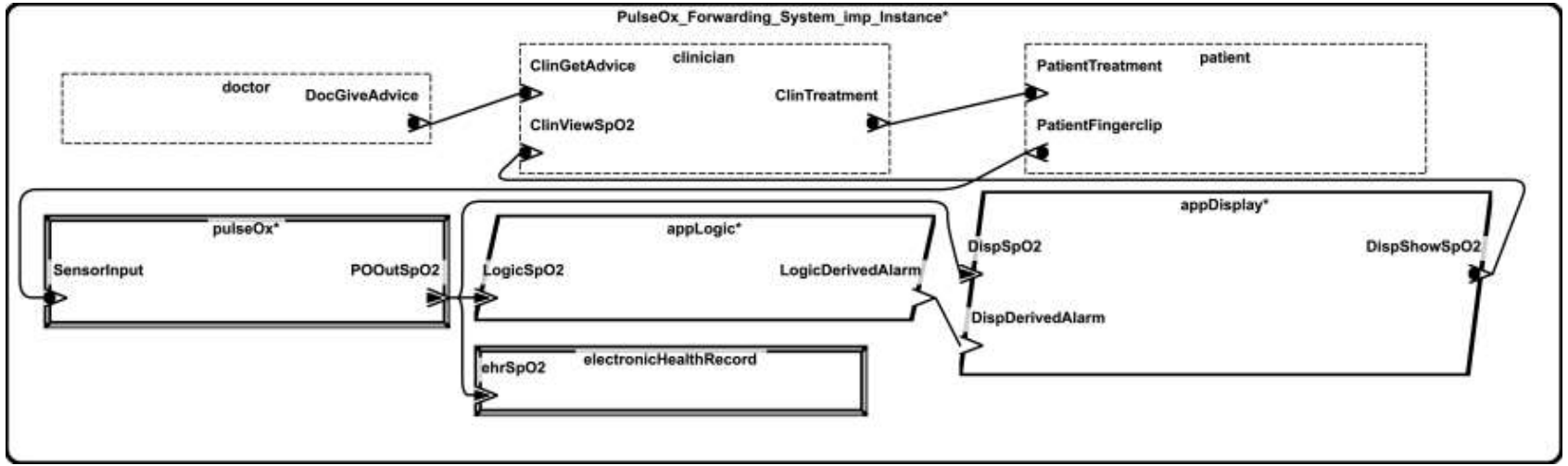
The PulseOx Forwarding Example



The PulseOx Forwarding Example

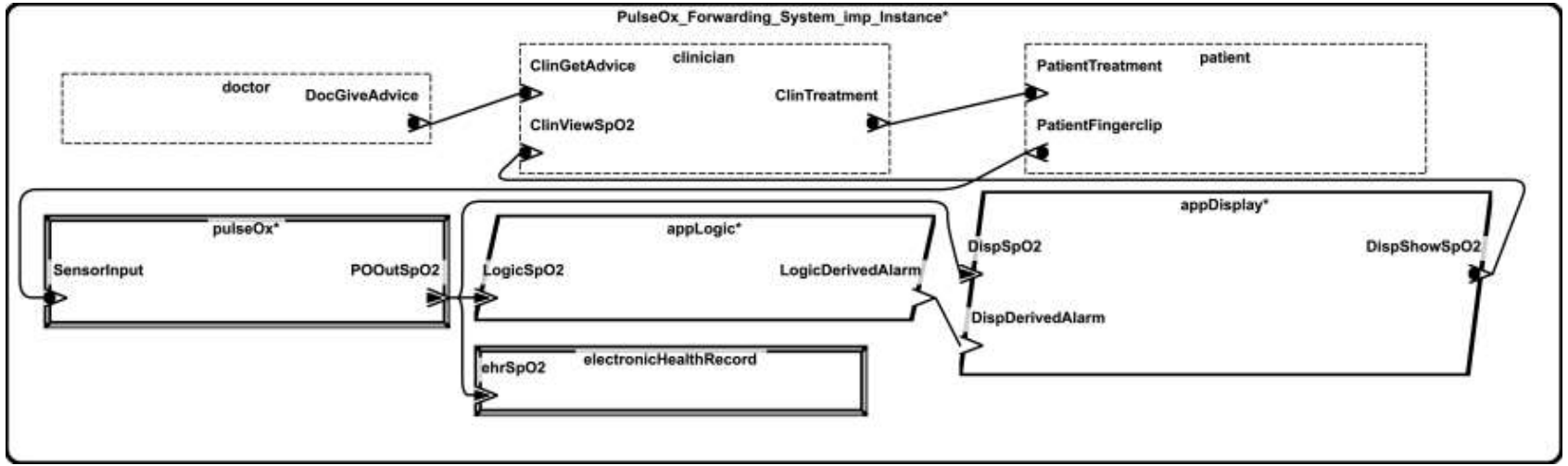


The PulseOx Forwarding Example



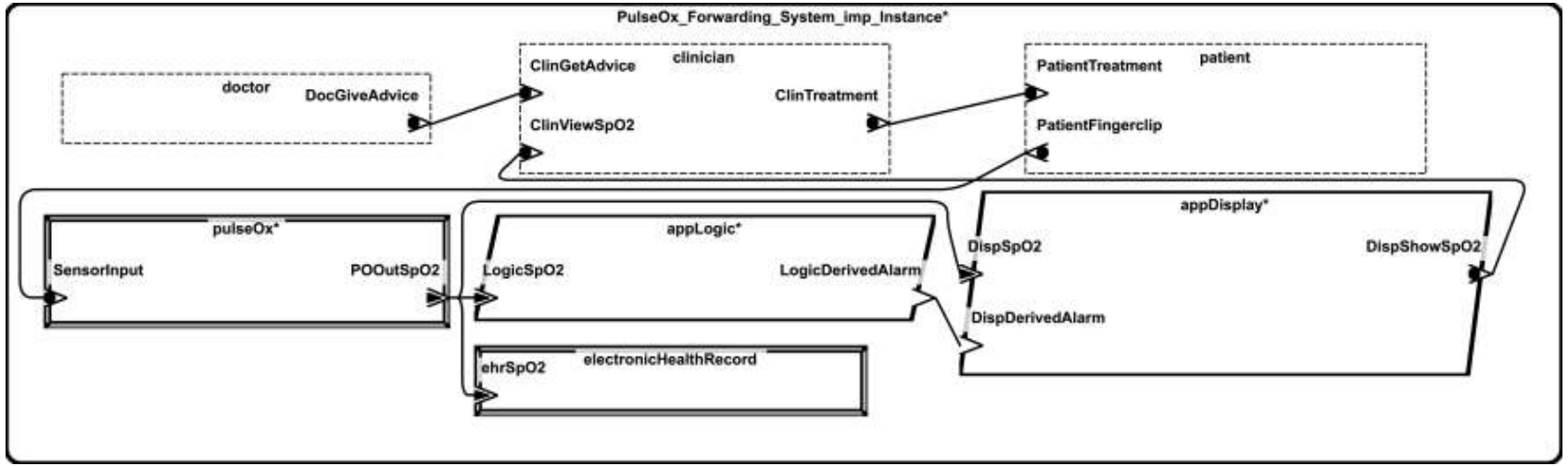
- Safety problem to avoid: Incorrect SpO_2 displayed

The PulseOx Forwarding Example



- Safety problem to avoid: Incorrect SpO_2 displayed

The PulseOx Forwarding Example



- Safety problem to avoid: Incorrect SpO_2 displayed
- AADL's "Error Modeling" (EMV2) annex can model these error propagations

STPA & SAFE

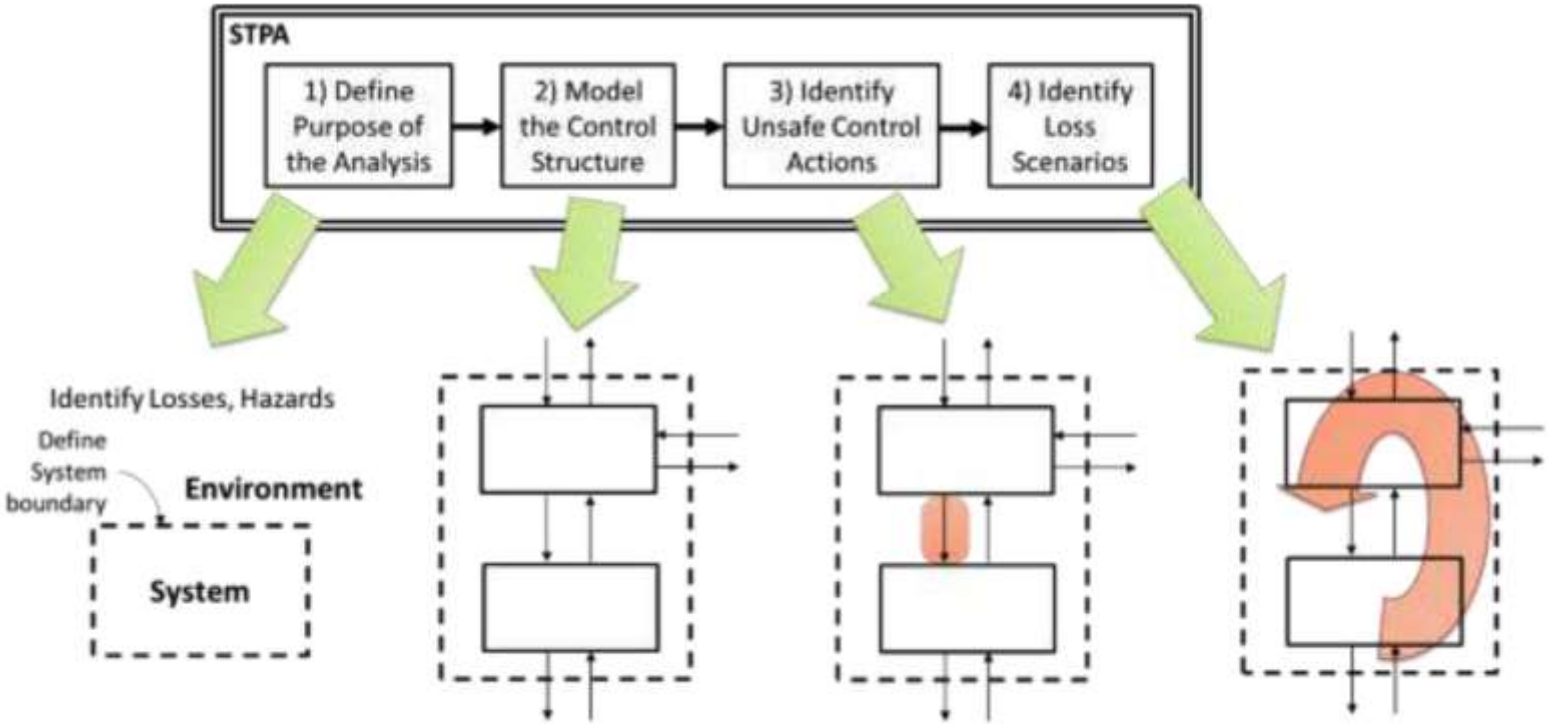


Figure 2.1: Overview of the basic STPA Method

© John Thomas, Nancy Leveson, STPA Handbook, March 2018
https://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf

Outline

1. Background
2. **ASAP: Three Viewpoints**
    1. Fundamentals (skipped for time)
    2. Connected Neighbors
    3. Unsafe Control Actions
3. Future Work

Viewpoint 2: Connected Neighbors

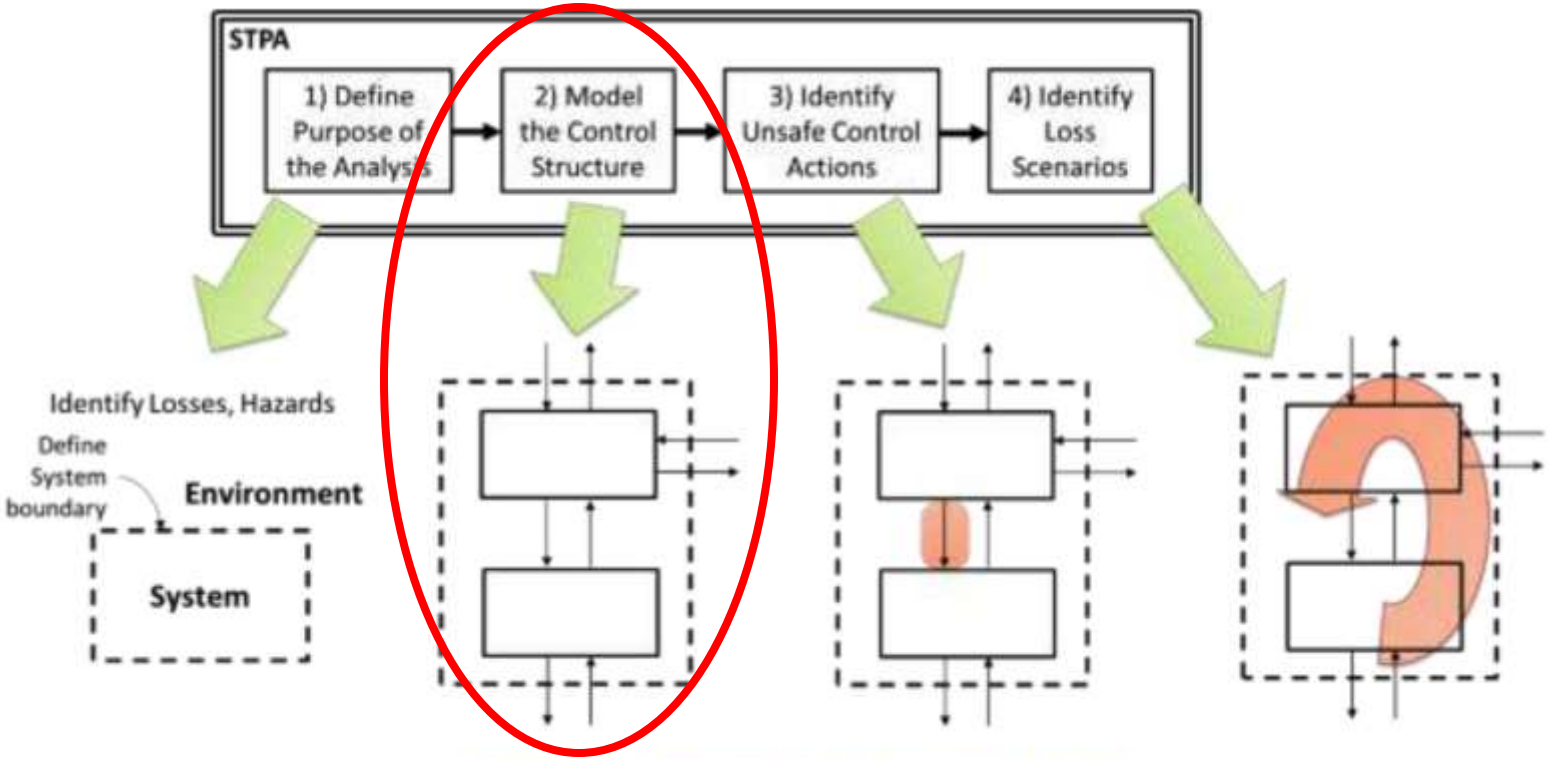
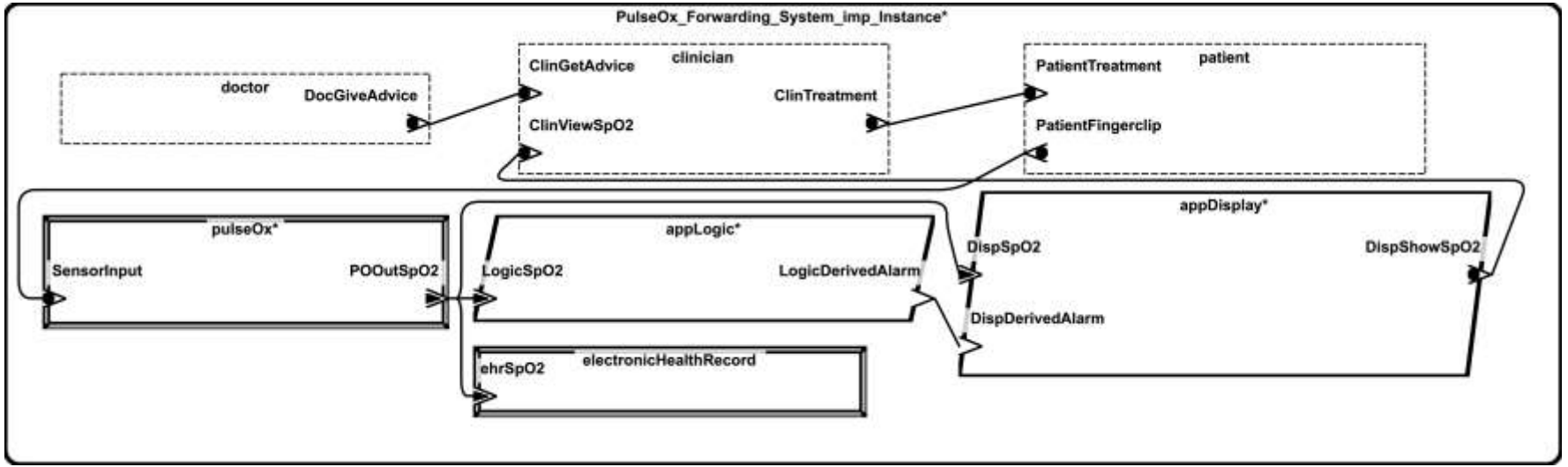


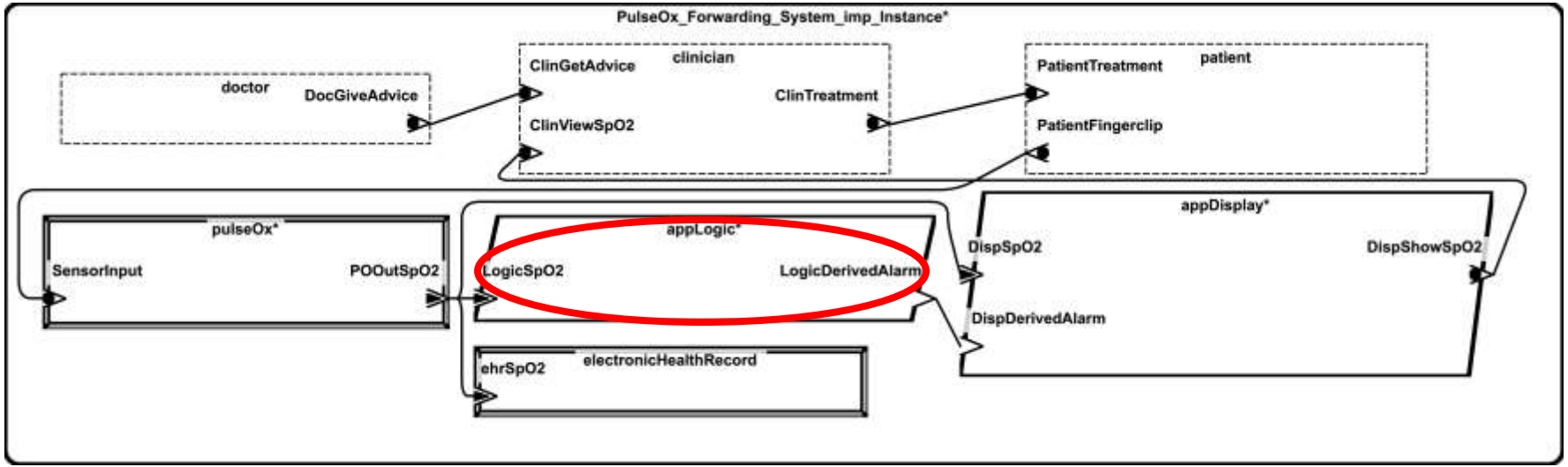
Figure 2.1: Overview of the basic STPA Method

© John Thomas, Nancy Leveson, STPA Handbook, March 2018
https://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf

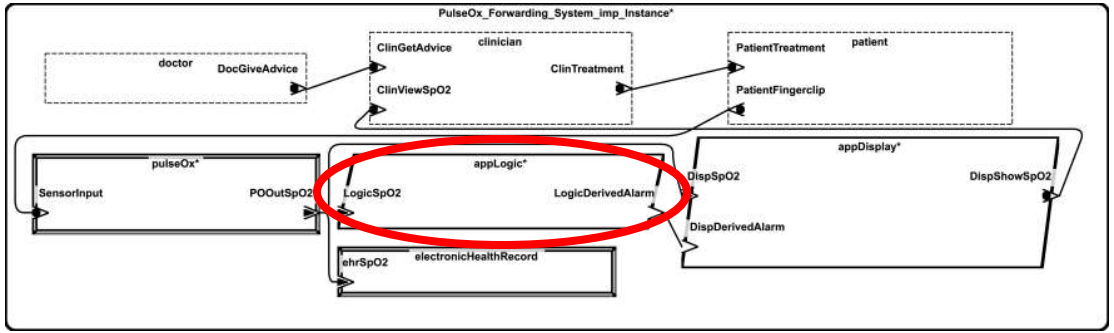
Viewpoint 2: Connected Neighbors

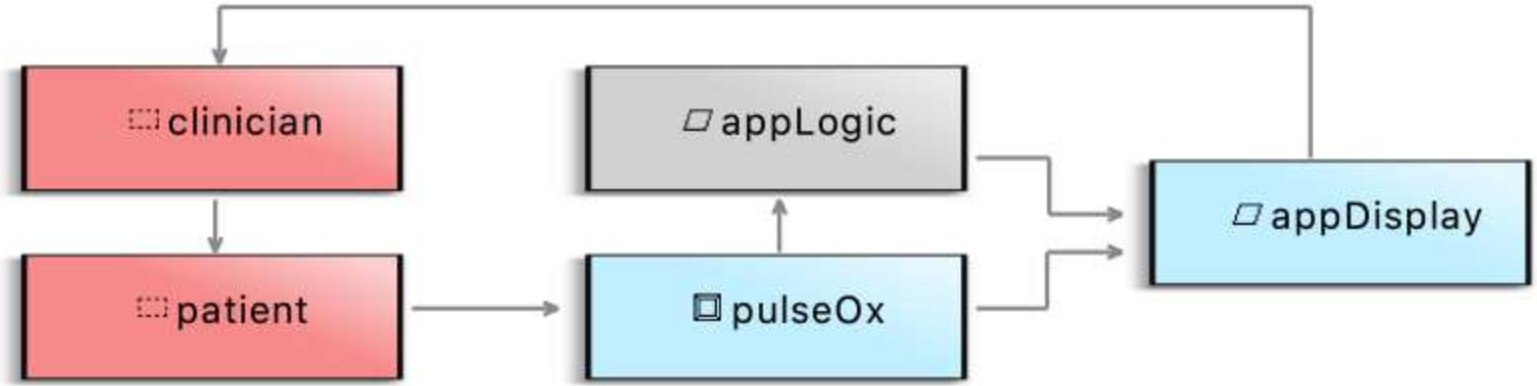


Viewpoint 2: Connected Neighbors



Viewpoint 2: Connected Neighbors





Viewpoint 3: Unsafe Control Actions

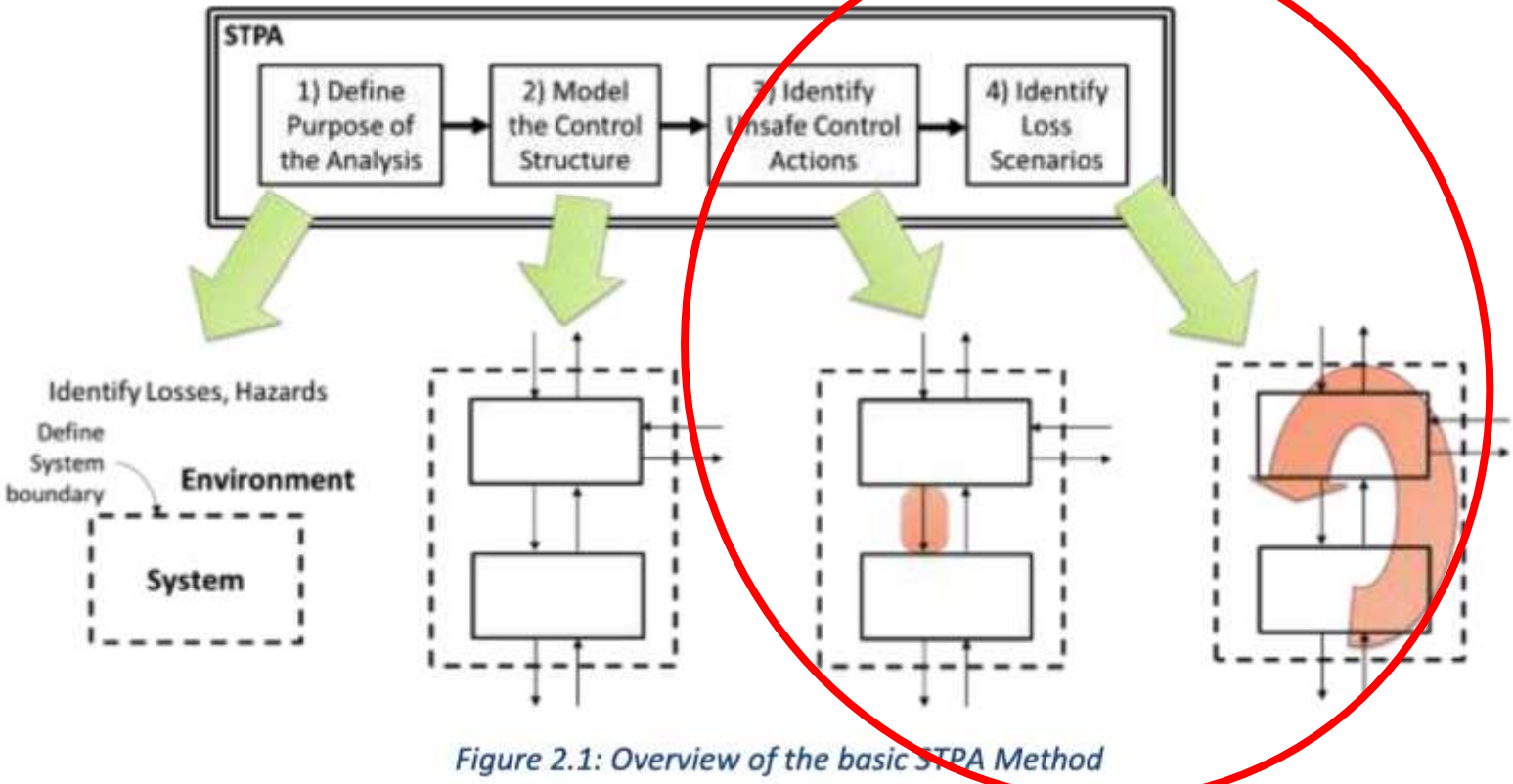


Figure 2.1: Overview of the basic STPA Method

© John Thomas, Nancy Leveson, STPA Handbook, March 2018
https://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf

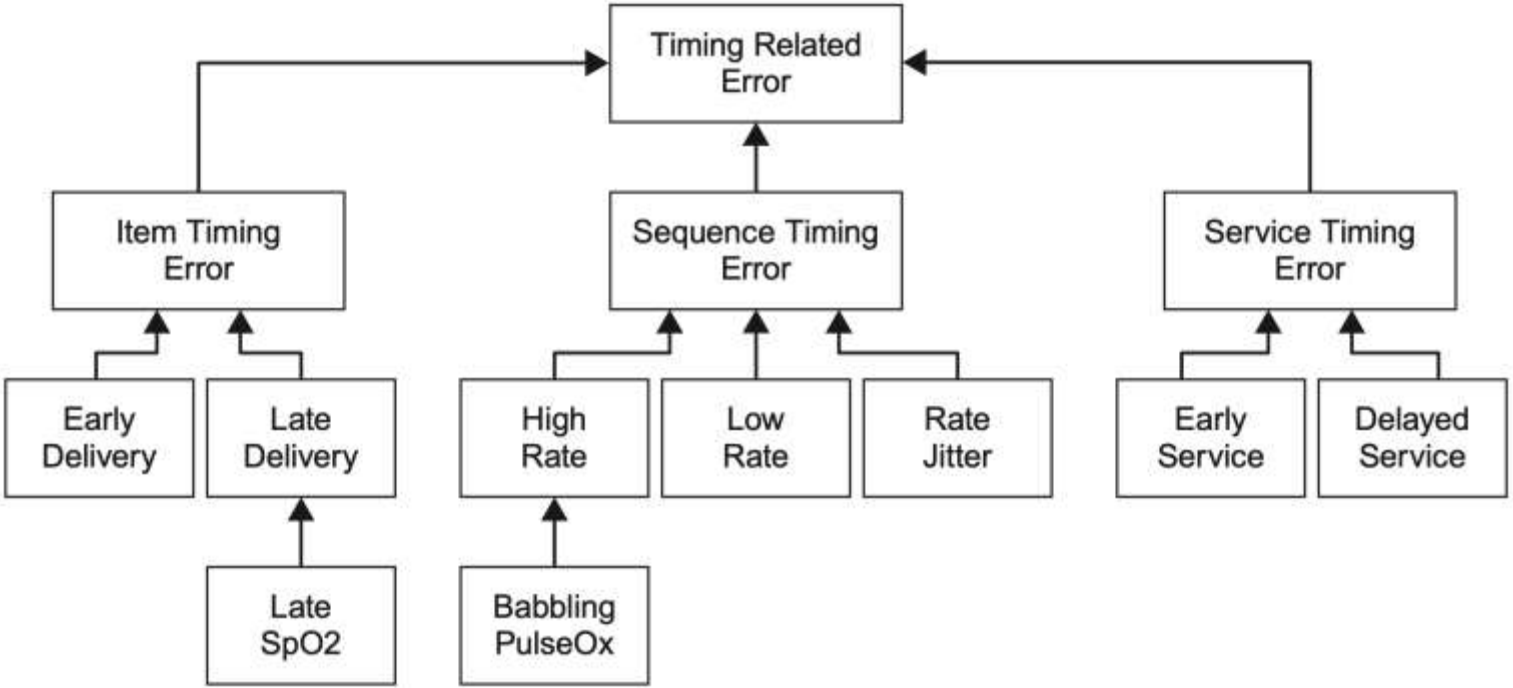
STPA's presentation of unsafe control actions

Control action (by Auto-Hold)	Not providing causes hazard	Providing causes hazard	Incorrect Timing/Order	Stopped Too Soon / Applied too long
Hold Command				
Release Command				

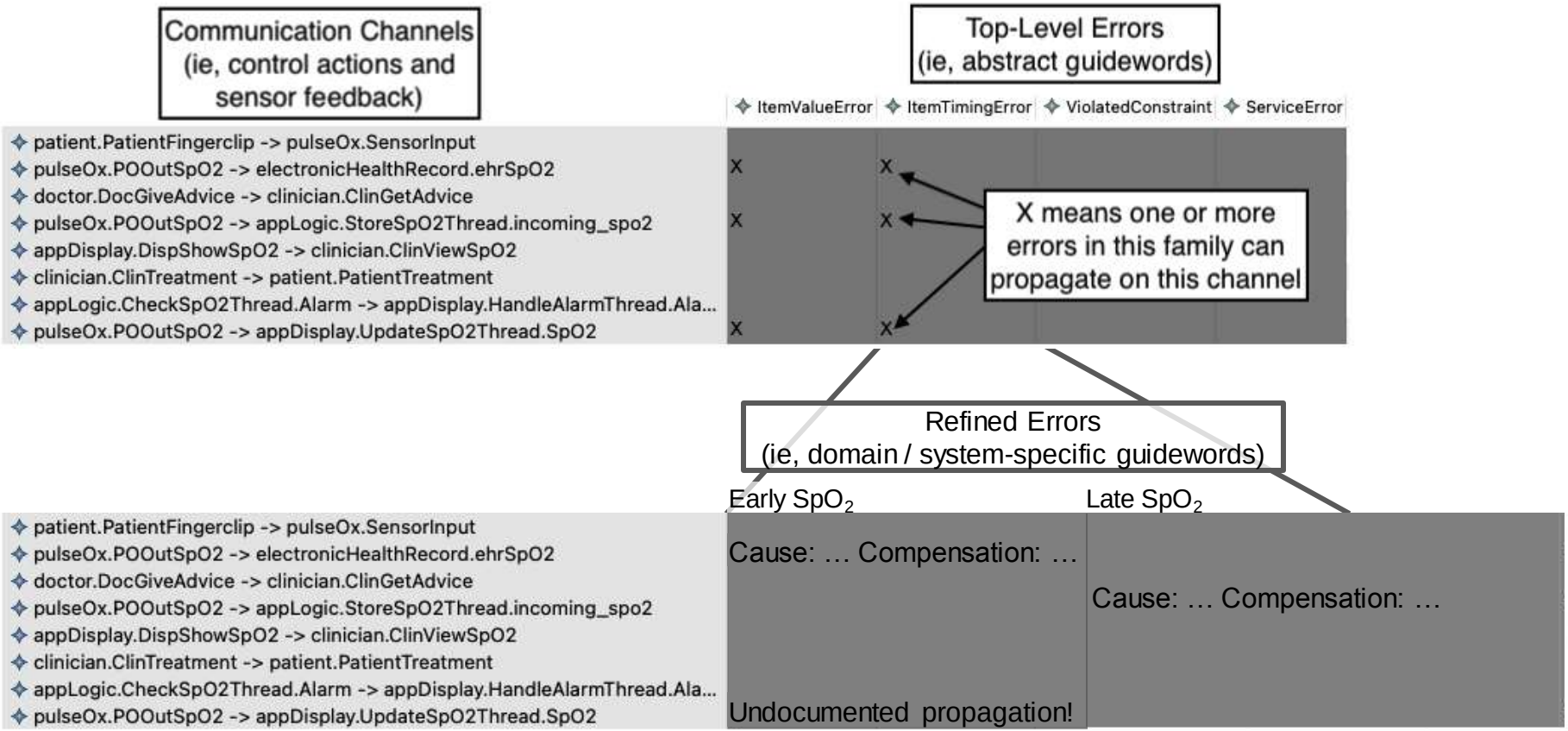
Adapted from John Thomas, Nancy Leveson, STPA Handbook, March 2018
https://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf

Viewpoint 3: Unsafe Control Actions

Interlude: The EMV2 Error Library



Viewpoint 3: Unsafe Control Actions



Outline

1. Background
2. ASAP: Three Viewpoints
3. **Future Work**

Future Work

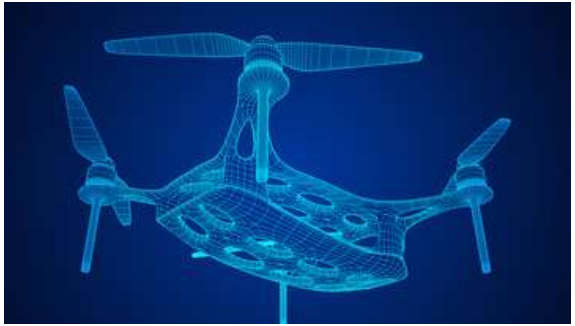
1. The “Focus” Action
2. Discovering accident causation

Outline

1. Introduction of the SAFIR project
2. Safety considerations for AI-enabled systems
3. Oqarina: mechanization of the AADL language using the Coq theorem prover
4. ASAP: the Architecture-Supported Audit Processor
5. Wrap-up discussion

LSI SAFIR Case Study – 6.2 variant

10.1109/ACCESS.2019.2909530



Let us consider a patrolling mission with the following:

- Partially known place: a factory, with slow-moving parts
- Both closed and open areas: wind, lighting conditions
- Find and detect intruders, recognize threats
- Tight maneuvers to enter/exit buildings, safety margins to avoid damages
- Autonomy in decision making, basic man-machine teaming
  1. classify threats, notification of threats to operator
  2. select next step: navigation plan / follow threat / back home for maintenance
  3. autonomous flight with switch to either human operated or fail-safe mode

FY21-22: theoretical foundations to support this case study

FY23: realization in a controlled environment with Georgia Tech