Safety-security co-engineering: formal outlook

Elena Troubitsyna (Assoc. Prof at Theoretical Computer Science, KTH)

Introduction

- Until recently the main focus of designing SCADA (supervisory control and data acquisition) systems has been on **safety**
  - Freedom of accidents due to system failure
- Fault tolerance: component faults do not result in a system failure
- Verification of software: unsafe states are not reached
- Closed systems:
  - “Not my job” attitude towards security

Introduction cnt.

- Increasing reliance on networking in modern SCADA systems
- Exploiting security vulnerabilities might result in loss of control and situation awareness and lead to safety-related hazards
  - Power outages, critical services unavailability, jeep hacking etc.

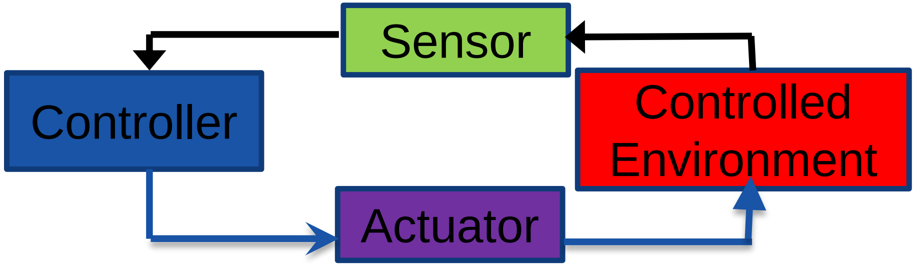
If not secure then not safe

How to achieve safety/security integration?

Motivation

- We need rigorous techniques that facilitate **systematic analysis of safety and security interdependencies and promote cyber-secure by construction system design**
- How to explicitly represent the impact of security failures and identify their impact on safety?
- Can we use models and associated proofs to identifying the security requirements derived from the system safety goals?
- Additional complexity: we need to consider both physical and cyber threats

Generic control system



Air-conditioning in this room

Sensor = temperature sensor

Actuator = heater

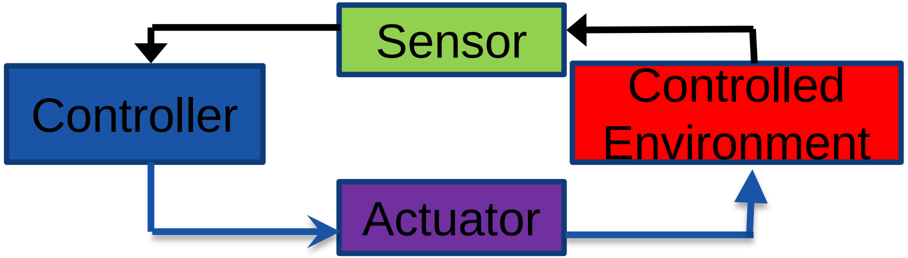
Control loop

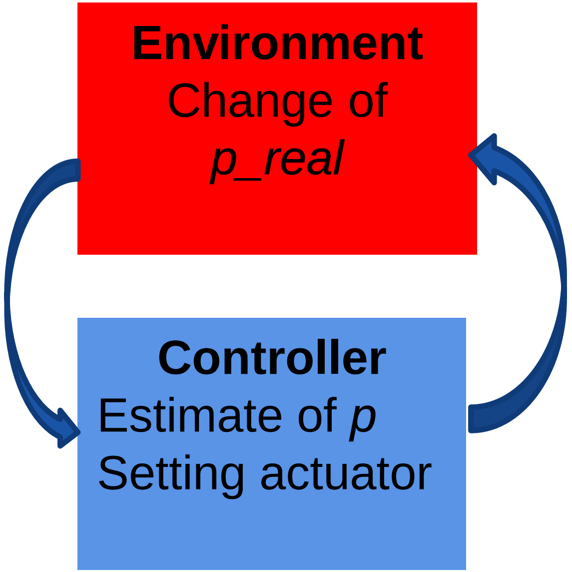
Read measurement of temperature sensor

If temperature > 24 degrees then heater : = OFF

If temperature < 22 degrees then heater := ON

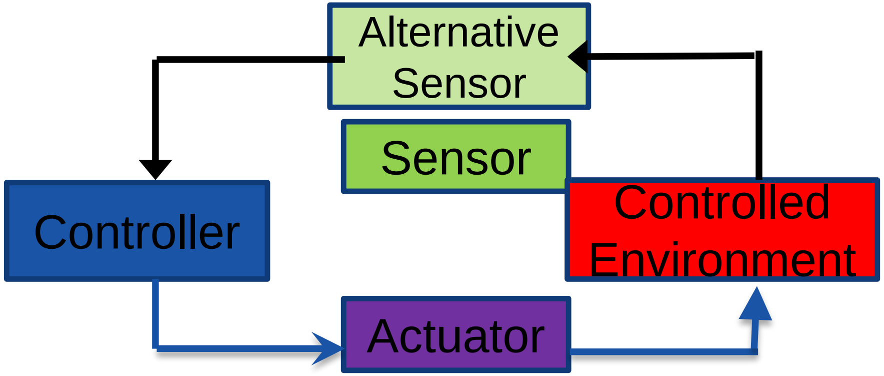
Generic control system

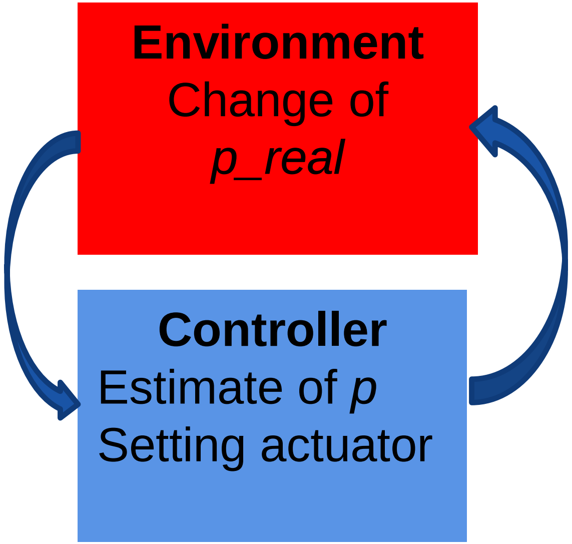




- Safety goal: keep safety parameter *p_real* within the predefined boundaries
- Safety invariant *p_crit_low* ≤ *p_real* ≤ *p_crit_high*

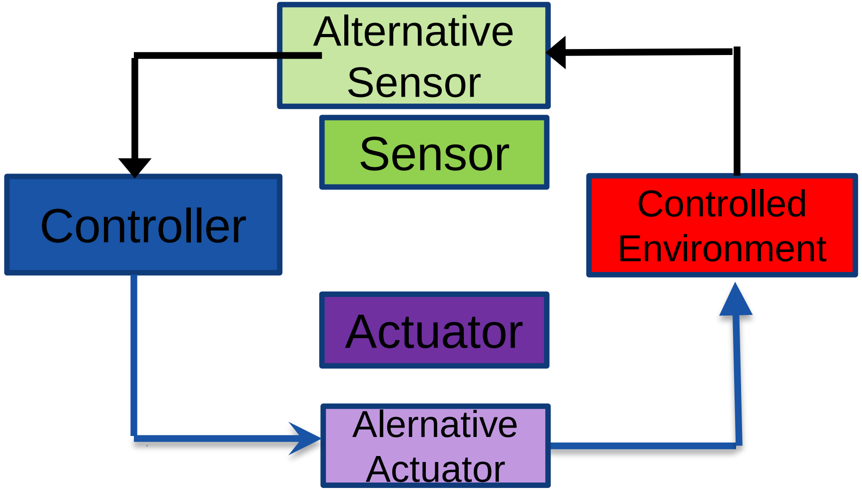
Generic control system

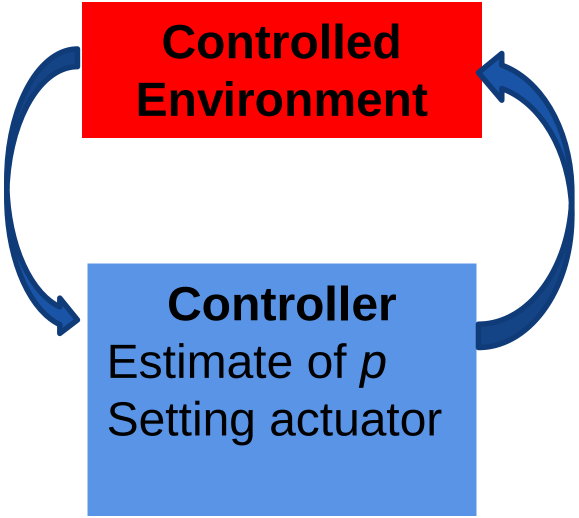




- Safety goal: keep safety parameter *p_real* within the predefined boundaries
- Safety invariant *p_crit_low* ≤ *p_real* ≤ *p_crit_high*

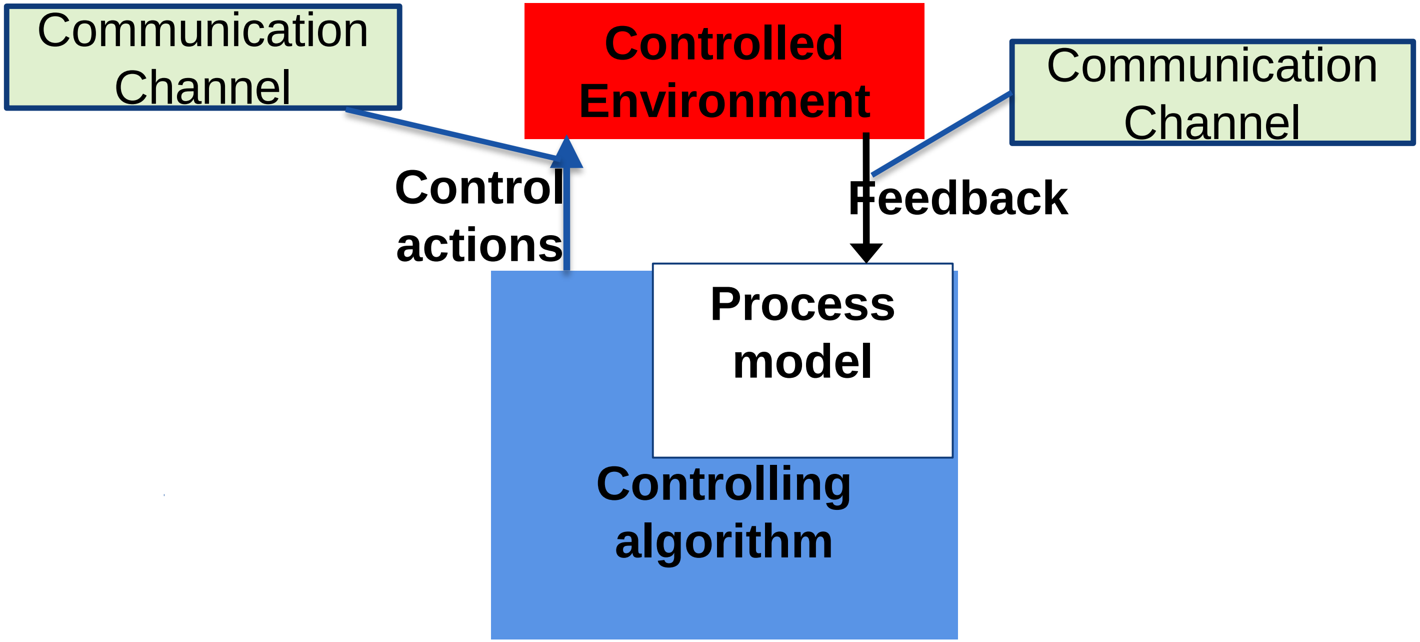
Generic control system



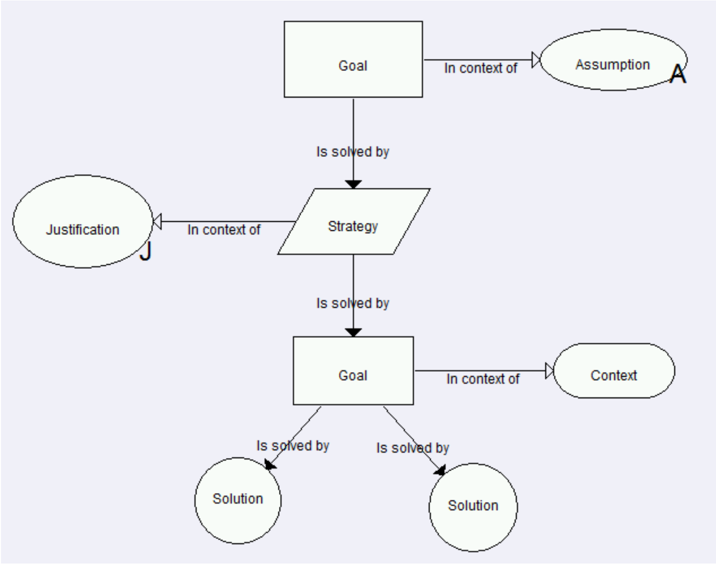


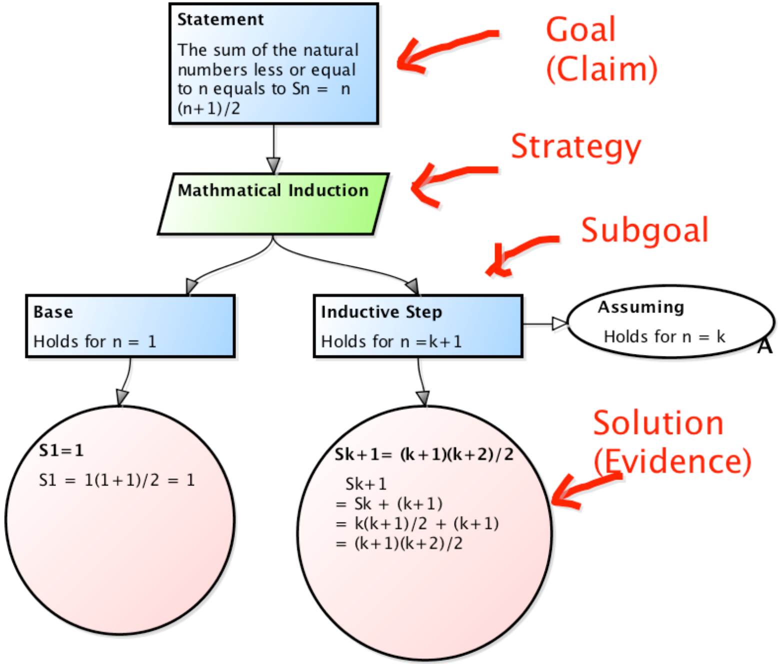
- Safety goal to keep safety parameter *p_real* within the predefined boundaries
- Safety invariant *p_crit_low* ≤ *p_real* ≤ *p_crit_high*

Control systems: systems-theoretic perspective

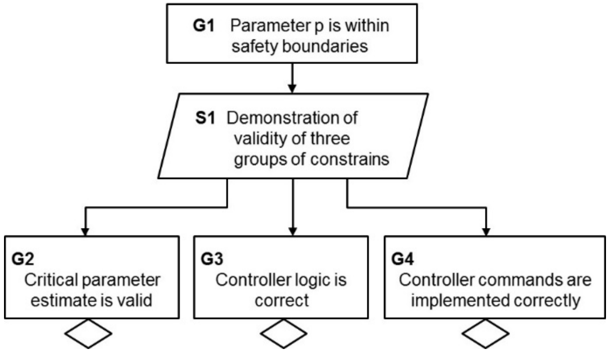


Safety cases





From safety case to cyber-security case



Constraint behind G2:
The value *p* used by the controller at each cycle as an estimate is sufficiently close to the real physical value *p_real* (Process model is sufficiently accurate)

Constraints behind G4:

- The actuator receives a command from the controller once per cycle (period)
- When the controller sets the actuator to the state decreasing then the value of *p_real* decreases (or stops increasing) with the passage of time, i.e.,

$$act = decreasing \Rightarrow p\_real_{c} \geq p\_real_{c+1}, \text{ for any system cycles } c \text{ and } c + 1$$

Decomposition of G3

Constraints behind G3

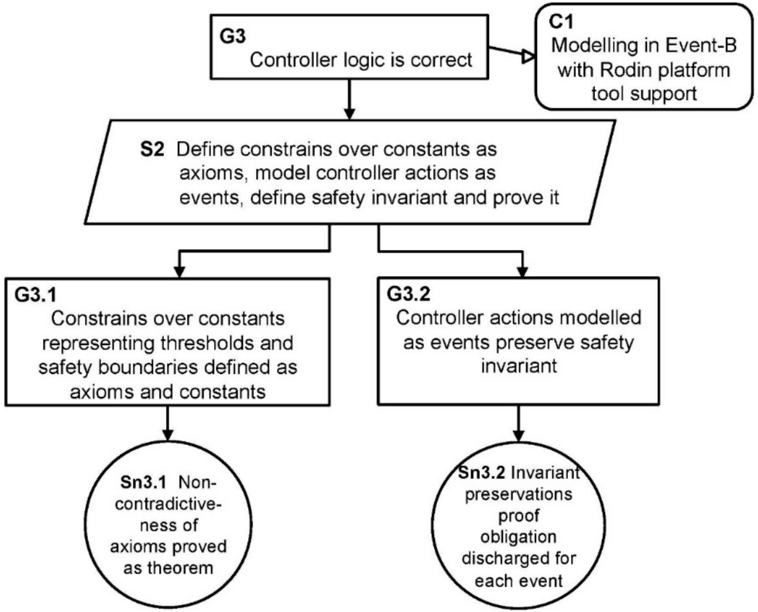
– Boundary *p_high* is calculated so that

$p\_high+\Delta+max\ \_increase\_per\ cycle \leq p\_crit\_high$;

– Effect of actuator state:

When *p* is greater than *p_high* then the controller always sets the actuator to the state decreasing

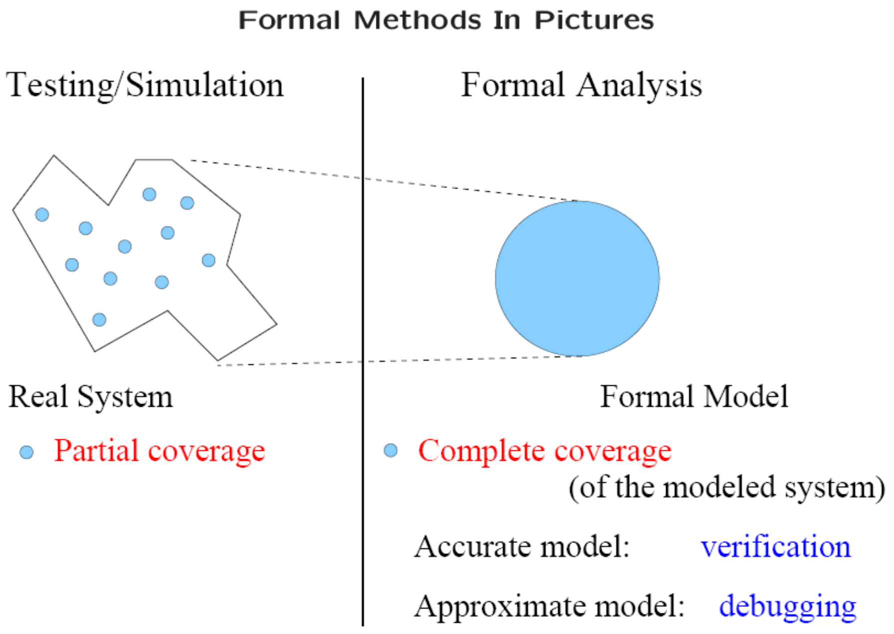
– Similarly to increasing



Formal specification and verification

- Formal specification languages:
  - mathematical description (specification) of high-level system requirements
- Specification has precise semantics
  - Verification tools allow us to prove that certain property is preserved
- Various generic and domain specific standards recommend the use of formal modelling in highly-critical systems
- Pros: find design errors before heavy investments in the implementation are made

Formal modelling in high assurance engineering



From J.Rushby talk on “Disappearing formal methods”

Event-B

- A state-based formal approach
- State is defined by a collection of variables
- Types of variables and properties are defined as invariants
- A context includes user-defined carrier sets, constants and their properties (defined as axioms)
- Dynamic behaviour is represented by events
- Model invariant defines a set of allowed (safe) states

Machine *M*
Variables *v*
Invariants *I*
Events
Initialisation
*evt*1
..
evtN

Context *C*
Carrier Sets *d*
Constants *c*
Axioms *A*

Event is a guarded command

stimulus → *response*

WHEN guard **THEN** assignment to variables **END**

Each event should preserve the invariant

Abstract specification of generic control system

Phases of control cycle

Actuator state

Controller estimate

Physical value

Controller estimates p

Setting actuator to decrease p

Setting actuator to increase p

Environment evolution

Machine Controller

Variables *phase, act, p, p_real*

Invariants $phase \in \mathrm{PHASE} \wedge act \in \mathrm{ACT} \wedge p \in \mathrm{N} \wedge p\_real \in \mathrm{N} \ldots \wedge \mathrm{safety}$

EVENTS

Initialisation $phase := est \;||\; act := none \;||\; p := p0 \;||\; p\_real := p\_real0$ end

estimate = where $phase = est$ then $p :\in estim(p\_real) \;||\; phase := cont$ end

act_decrease = where $phase = cont \wedge p \geq p\_high$ then $act := decreasing \;||$

$phase := env$ end

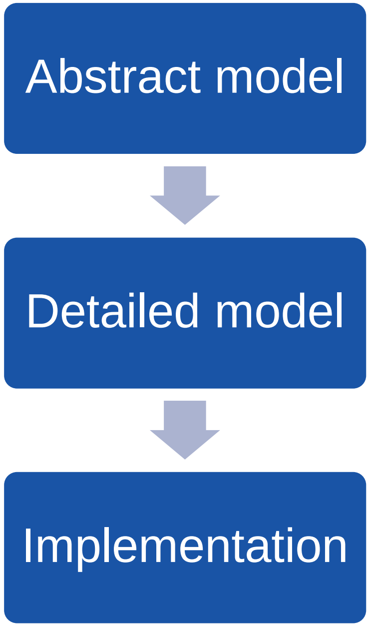
act_inc = where $phase = cont \wedge p \leq p\_low$ then $act := increasing \;||$

$phase := env$ end ...

env = **where** $phase = env$ **then** $p\_real :\in \mathrm{N} \;||\; phase := est$ **end**

Correct-(and dependable)-by-construction development in Event-B

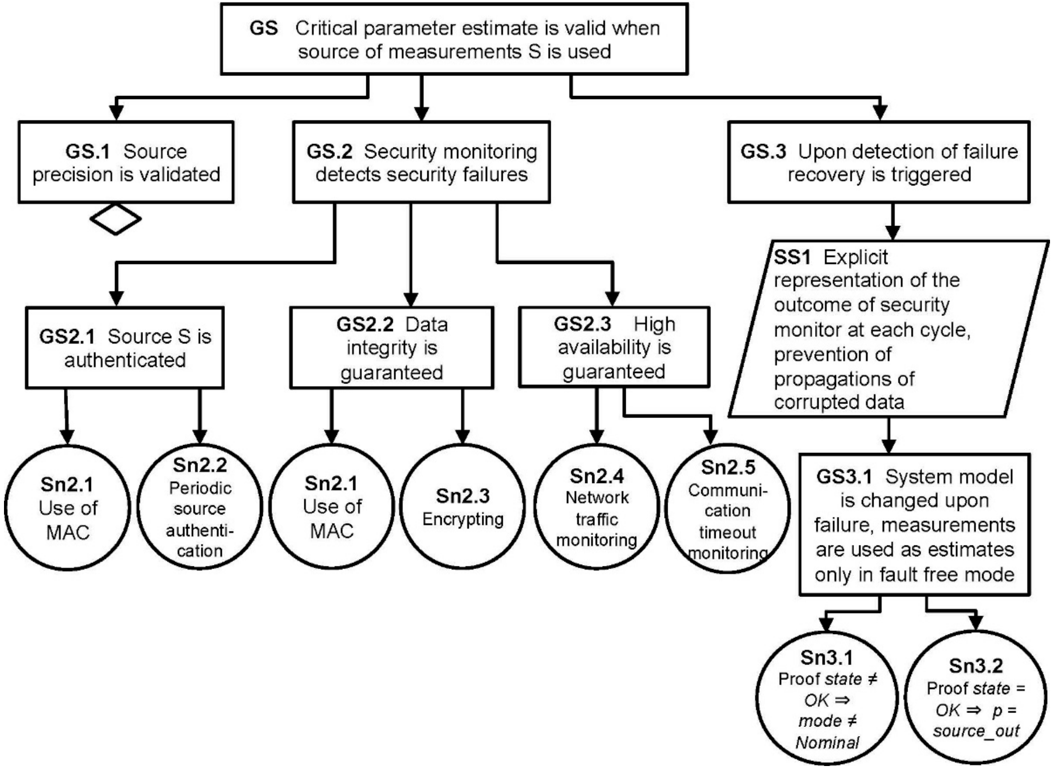
- Abstract model: “birds view” – defines only the most essential properties and behavior
- Refinement model transformation: more detailed requirements and properties are added
- Correctness of model transformation is proved: correspondence between more abstract and more concrete state spaces implies that abstract invariant is preserved in the refined model
- Explicit representation of dependability features: safety, fault tolerance, adaptability
- Rodin platform: automated support for model construction and verification: (incremental development merging modelling and verification)



Constructing specification and cyber-security case

- Incremental derivation of the networked architecture by refinement in parallel with safety case
- We unfold the system architecture together with explicit specification of communication links by model refinement.
- Data producer-consumer pattern: abstraction of the impact of the security failures
  - > *spoofing producer*
  - > *data tampering*
  - > *DOS (channel unavailability)*
- We introduce a model of the sensor and sensor-actuator comm.link (producer: sensor, consumer: controller)
- Derived constraints:
  - – sensor imprecision is acceptable ($\leq \Delta$)
  - – controller does not use corrupted data as an estimate of p
  - – detection of a corrupted value triggers error recovery and activates an alternative mode of estimating p.

Corresponding fragment of safety case



Conclusions

- Systems theoretic approach provides us with a suitable basis for an integrated analysis of safety-security requirements
- Modelling allows us to treat safety and security as the interdependent constraints
  - Enables identification of the critical paths including reconfiguration
- Derived constraints are heterogeneous: sw, hw, system design
- Current work: quantitative security analysis – likelihood of attack success for various attacker profiles and model-based evaluation of protection alternatives

Thank you!