Assurance-Driven Design of Cyber-Physical Systems[1]

N. Shankar

Computer Science Laboratory
SRI International
Menlo Park, CA

July 16, 2014

[1] Supported by NASA NRA NNA13AC55C, NSF Grant CNS-0917375, and DARPA under agreement number FA8750-12-C-0284. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NASA, NSF, DARPA, or the U.S. Government.

Motivation

- An assurance case is a *"a documented body of evidence that provides a convincing and valid argument that a specified set of critical claims about a system's properties are adequately justified for a given application in a given environment."* [Adelard]
- From the FDA Draft Guidance document *Total Product Life Cycle: Infusion Pump - Premarket Notification [510(k)] Submissions*:

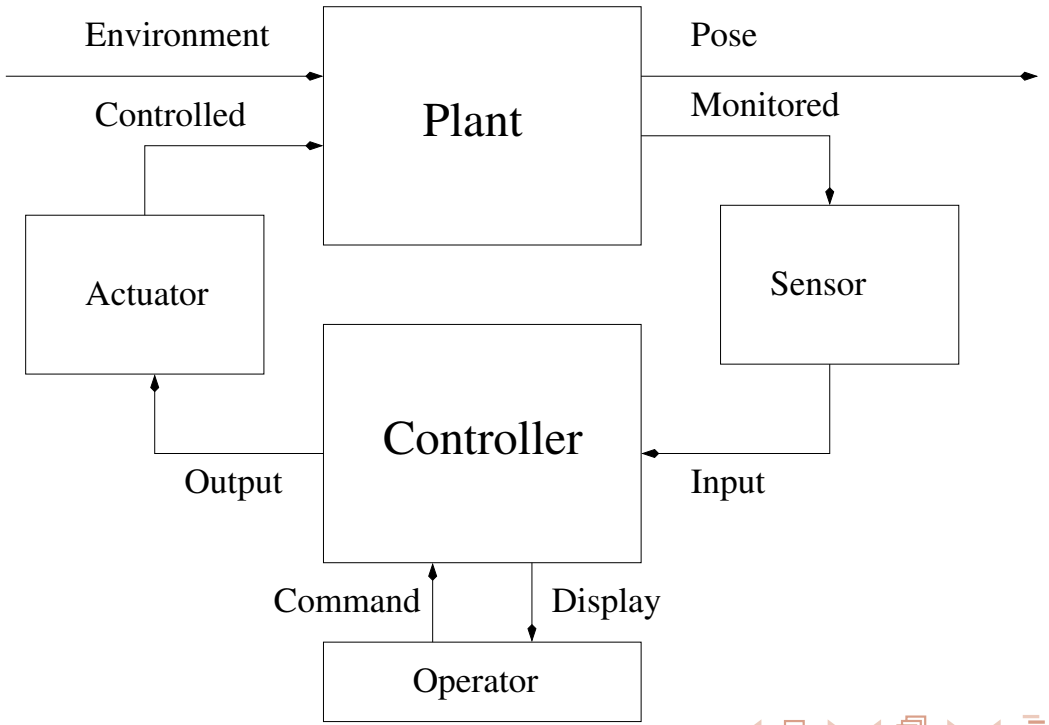
  An assurance case is a formal method for demonstrating the validity of a claim by providing a convincing argument together with supporting evidence. It is a way to structure arguments to help ensure that top-level claims are credible and supported. In an assurance case, many arguments, with their supporting evidence, may be grouped under one top- level claim. For a complex case, there may be a complex web of arguments and sub-claims.

Talk Outline

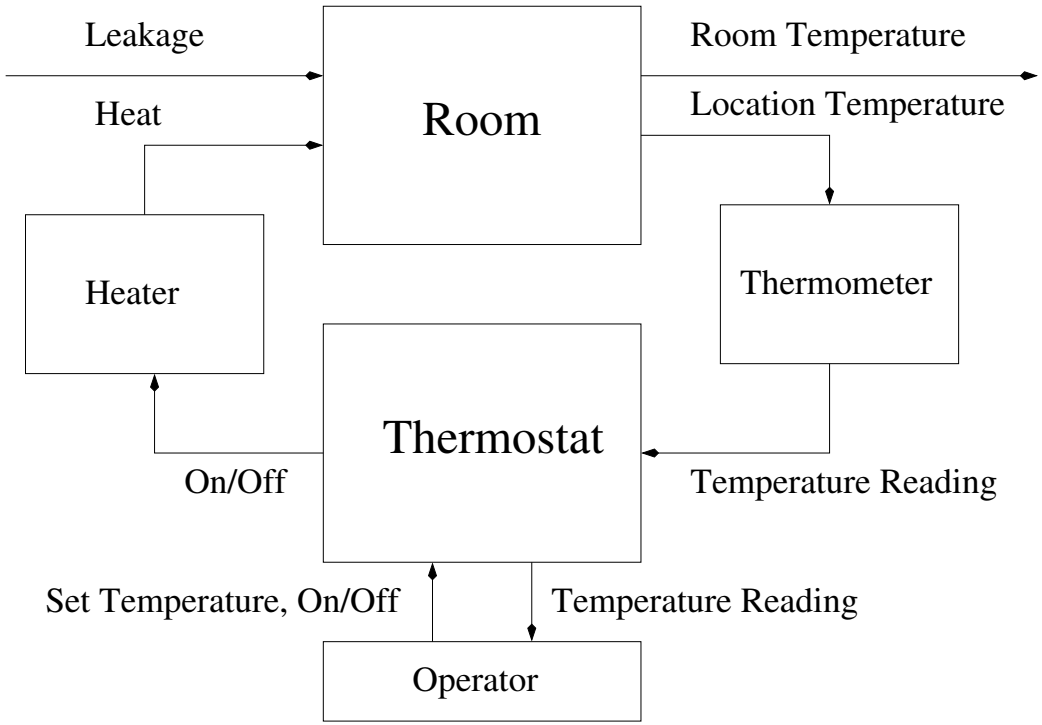
- Key Challenges are
  - How do we systematically construct assurance cases consisting of claims, evidence, and arguments (formal, semi-formal, and informal)?
  - Can we make assurance an integral goal of the design process?
- Assurance for cyber-physical systems
  - Eight-variables model
  - Layered assurance of cyber-physical systems: The Landshark Example
- Architecture for cyber-physical systems
  - Robot Architecture Definition Language (RADL)
  - The Quasi-synchronous model of computation
- The Evidential Tool Bus
  - What is the Evidential Tool Bus?
  - How is it used in defining assurance workflows and arguments?

Cyber-Physical Systems: Eight Variables Model

- These are systems composed of physical and computational components, with multiple control loops operating at multiple time scales.
- CP stems are typically distributed and consist of a network of sensors, controllers, and actuators.
- The whole system can itself be seen as a giant control loop with a plant and a controller.

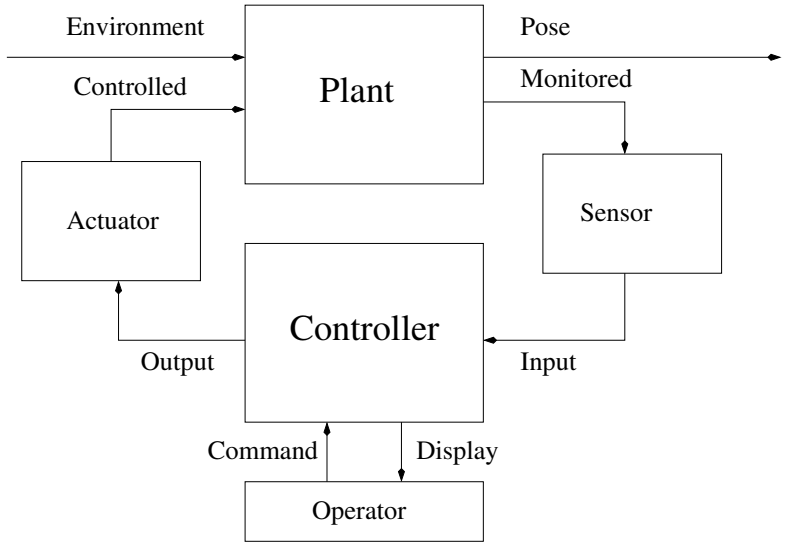


A Simple Example: Room-Heating Thermostat



1. The *plant* consists of the room whose temperature is being maintained, the *actuator* is the heater, and the *environment* is the energy leakage from the room.
2. The goal *requirement* is to maintain the average temperature across the room above a specified temperature that is set by the operator.

Top Assurance Claim




```
  EnvironmentAssumption(environment) AND
  PlantModel(environment, control, pose, monitor) AND
  SensorAccuracy(monitor, input) AND
  ActuatorResponse(output, control) AND
  ControllerOutput(input, command, output, display) AND
  OperatorModel(display, command)
IMPLIES
 Requirement(command, environment, pose, display)
```

The Refinement Layers in the Assurance Argument

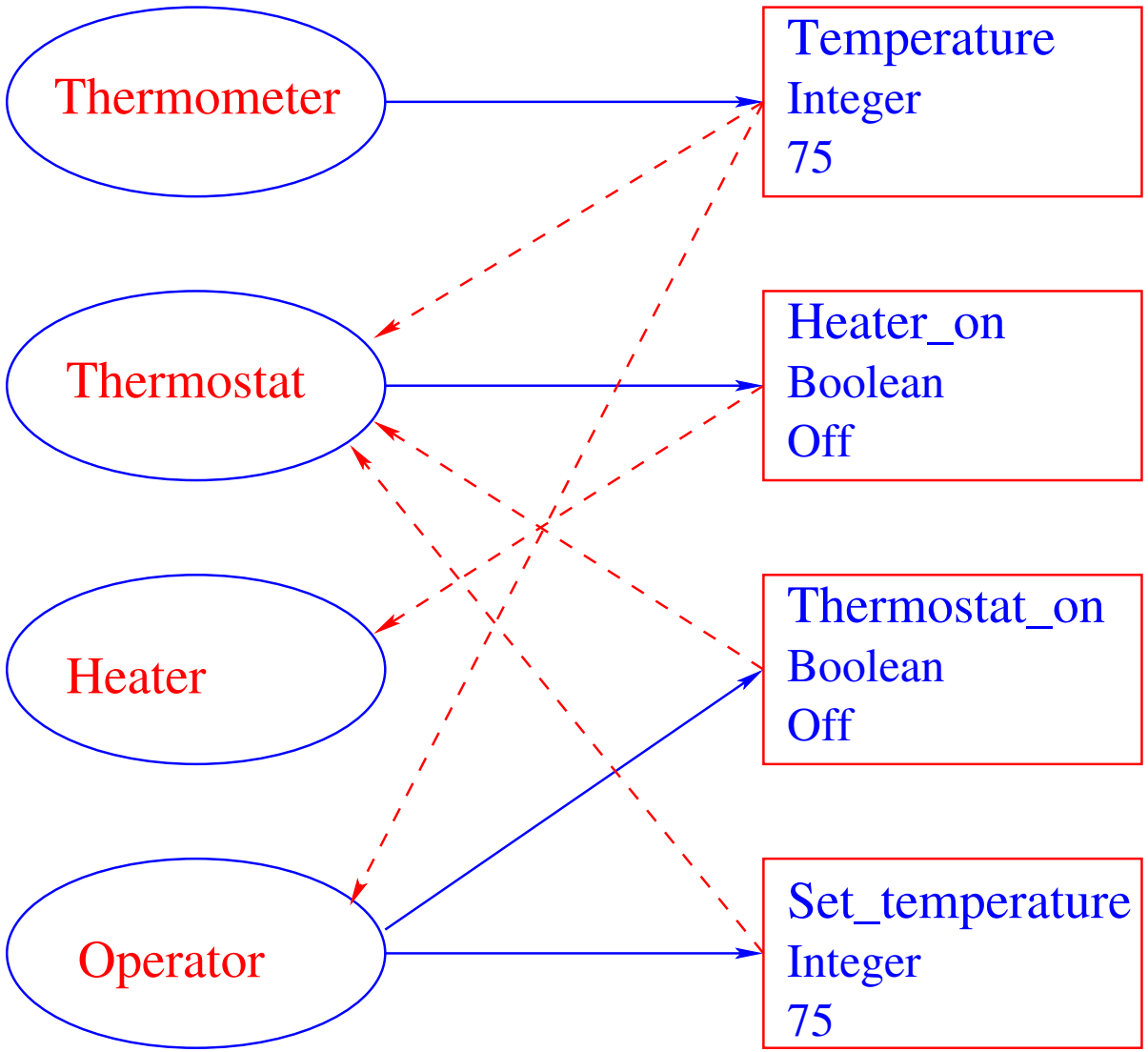
- The argument is structured into three refinement layers where each layer is shown to implement the assumptions imposed by the higher layer:
  1. **The Mathematical Model:** A spatio-temporal model that captures the physics of the vehicle, the environment assumptions, the system-level requirements, and the mathematical designs of the controllers and monitors.
  2. **The Engineering Model:** Algorithmic/architectural models for plants and controllers/monitors, fault models for the physical components, and a model of computation (MoC) for communication and computation.
  3. **The Computation Model:** The software from the engineering model are executed as processes within a hypervisor partition and communicating using hypervisor/network services.
- Each layer also introduces fault models and mitigations for the components relevant to it.

Robot Architecture Definition Language (RADL)

The Robot ADL bridges the gap between the engineering and computation models.
The ADL is inspired by the popular Robot Operating System (ROS) middleware for building cyber-physical systems.
The architecture definition captures

1. Message types
2. Nodes, with their period, initialization, and steady-state computation steps, published topics, received topics (with latency bounds), and devices
3. Topics, with a message type, period, and authentication
4. Mapping of nodes to partitions within processors and associated devices
5. Mapping of channels to buses with firewalls and authentication

Thermostat in ROS



RADL: Nodes

```
thermostat : node {
    SUBSCRIBES
        setting { TOPIC thermostat_setting
                  SUBSCRIBER default_sub MAXLATENCY msec 1 }
        thermom { TOPIC thermometer_data
                  SUBSCRIBER default_sub MAXLATENCY msec 1 }
    PUBLISHES
        control { TOPIC thermostat_control PUBLISHER default_pub
    CXX {
        CLASS "Thermostat" HEADER "thermostat.h"
        FILENAME "thermostat.cpp" }
    PERIOD msec [49, 51]
    WCET msec 2
}
```

A RADL Device Node

```
thermometer : node {
    PUBLISHES
        temp { TOPIC thermometer_data PUBLISHER default_pub }
    CXX { CLASS "Thermometer"
        HEADER "thermometer.h" FILENAME "thermometer.cpp"}
    DEVICES
        temperature_sensor
    PERIOD msec [49,51]
    WCET msec 2
    }
```

RADL: Topics

```
thermostat_set : topic {
    FIELDS
        temp : float32 75
}
```

RADL: Platform

```
blackbox: PROCESSOR {
  DEVICE
   thermometer1: thermometer
   heater1: heater
   radio1: radio
  HYPERVISOR
   CertiKOS
  PARTITIONS
   controllerPartition
   heaterPartition
   thermometerPartition
  BUSES
   CertiKOS_IPC
   USB
   Ethernet
  }
```

RADL: Mappings

```
controllerPartition: PARTITION {
  OS Ubuntu 12.0.4
  PACKAGES
    ...
  NODES
   thermostat
}
```

RADL: Buses

```
CertiKOS_IPC: BUS {
  ENDPOINTS
    controllerPartition
    thermometerPartition
    heaterPartition
}

Ethernet: BUS {
  ENDPOINTS
    consolePartition
    controllerPartition
}
```

Using RADL

- The architecture description is used to
  1. Check the architectural integrity to ensure that the message types, rates, and device assignments match
  2. Relate the model of computation at the engineering layer to the ROS configuration and platform in the computation layer
  3. Generate glue code to handle initialization, and access to devices and communication
  4. Build and certify the running ROS configuration to conform to the architecture
- The model(s) of computation in the engineering layer is correctly realized
  1. The system is properly initialized
  2. The nodes are scheduled at their specified periodic rate and meet their WCET.
  3. The node computations are executed correctly within the hypervisor
  4. The messages are delivered with integrity and authenticity within the latency bounds
  5. The fault assumptions in the model are met by the architecture

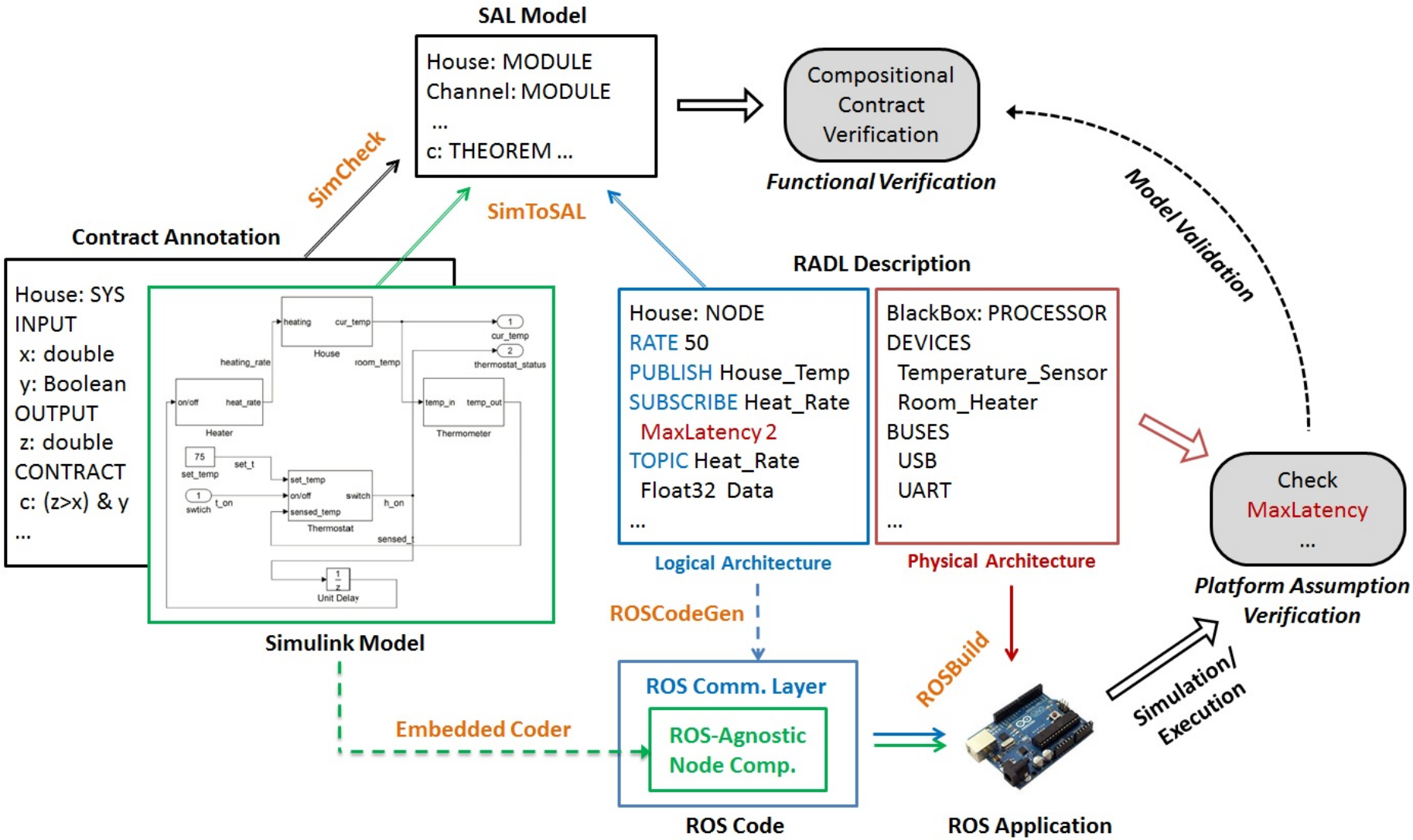
Semantics of RADL

- Semantics is captured by a PVS theory `RADLsemantics` with uninterpreted types for `topic`, `node`, `data`, `flag`, and `nodestate`.
- A `message` contains `data` and `flag`.
- An `event` contains `message` and `timestamp`.
- A stream is a sequence of events with time that is nondecreasing and eventually exceeding any bound.
- A node event consists of a `nodestate` and a `timestamp`.
- A node computation is a sequence of node events with same constraints on time as above.
- The outputs for each node event are published within the WCET of the timestamp.

The Quasi-Synchronous Model of Computation

- Synchronous models has been heavily studied, but they are impractical for typical distributed control systems.
- The quasi-synchronous model consists of locally clocked processes, but with a bounded drift between the rates of individual clocks.
- Several basic theorems can be proved (in PVS) about this model:
  1. The latency for processing a transmitted message is bounded by the maximum transmission latency and the maximum receiver period.
  2. Under some assumptions on the relative rates and drifts, the number of consecutive dropped (unprocessed) messages is bounded
  3. With added assumptions on the queue length, no messages are dropped.
  4. The staleness of a processed message is bounded
- These theorems are crucial for proving physical system properties.

RADL Assurance Flow



Physical Architecture

- The RADL physical architecture assigns nodes to partitions (VMs) running on a hypervisor executing on a (possibly multi-core) processor.
- Topics are assigned to bus channels, including inter-process communication (IPC) implemented by the hypervisor.
- The physical architecture must implement the period, latency, and WCET assumptions in the logical architecture.
- The middleware — ROS in this case — manages the application-specific tasks like the network initialization, node scheduling, the socket interface including marshalling/unmarshalling of data, authentication, and hypercalls.
- The hypervisor ensures memory partitioning, and implements timely and authenticated inter-process communication, and timely inter-processor communication.

ETB Overview

- The Evidential Tool Bus (ETB) is a distributed tool integration framework for constructing and maintaining claims supported by arguments based on evidence.
- ETB provides the infrastructure for
  - Creating workflows that integrate multiple tools, e.g., static analyzers, dynamic analyzers, satisfiability solvers, model checkers, and theorem provers
  - Generating claims based on these workflows
  - Producing checkable evidence (e.g., files) supporting these claims
  - Maintaining the evidence against changes to the inputs
- ETB is implemented in Python 2.7.

Datalog as a Metalanguage for ETB

- Datalog is a fragment of Horn-clause logic programming first introduced in the 1970s as a database query language.
- A Horn clause is of the form $P :- Q_1, \ldots, Q_n$, where ':-' can be read as 'if'.
- Query languages based on first-order logic can't represent recursive queries like transitive closure (*parent* is a database relation):

```
ancestor(x, y) :- parent(x, y)
ancestor(x, y) :- parent(x, z), ancestor(z, y)
```

- In ETB, we admit interpreted predicates that are evaluated using wrappers, e.g., the evaluation of `veryComposite`$(8, 3)$, can dynamically generate the clause

$$\begin{aligned} \mathit{veryComposite}(8,3) \quad :- \quad & \mathit{composite}(8), \\ & \mathit{composite}(9), \\ & \mathit{composite}(10). \end{aligned}$$

DO-178C Certification Argument in ETB

- The top-level claim is that the assurance artifacts `AA` are DO-178C compliant.
- A Datalog rule decomposes the claim into several subclaims.

```
do178c_compliant(AA) :-
  do178c_a2_check(AA),
  do178c_a3_check(AA),
  do178c_a4_check(AA),
  do178c_a5_check(AA),
  do178c_a6_check(AA),
  do178c_a7_check(AA).
```

The Certification Flow

- The verification workflow is launched before the compliance check

```
do178c_verify(AA) :-
     a2verify(AA, A2Evidence),
     a3verify(AA, A3Evidence),
     a4verify(AA, A4Evidence),
     a5verify(AA, A5Evidence),
     a6verify(AA, A6Evidence),
     a7verify(AA, A7Evidence).
```

DO-178C A4: Verification of Outputs of Software Design Process

```
a4verify(AA, A4Evidence) :-
          a4artifacts(AA, HLR, LLR, SWA, A4Checklist, ReviewDocs),
          a4checklist_reviewed(A4Checklist, ReviewResult),
          a4verify_checklist(HLR, LLR, SWA, A4Checklist,
                                        ..., A4Evidence).
```

a4verify_against_checklist is actually a wrapper that generates the clause below, which triggers the tool wrappers.

```
a4verify_checklist(HLR, LLR, SWA, A4Checklist, ..., A4Evidence) :-
    design_llr_hlr_compliance(LLR, HLR, VerificationReport_1),
    design_model_trace_anchor(LLR, HLR, VerificationReport_2),
    ...,
    design_llr_conforms_to_standards(LLR, VerificationReport_5)
    ...
    package_a4evidence(..., VerificationReport_5, ..., A4Evidence)
```

The Certification Phase

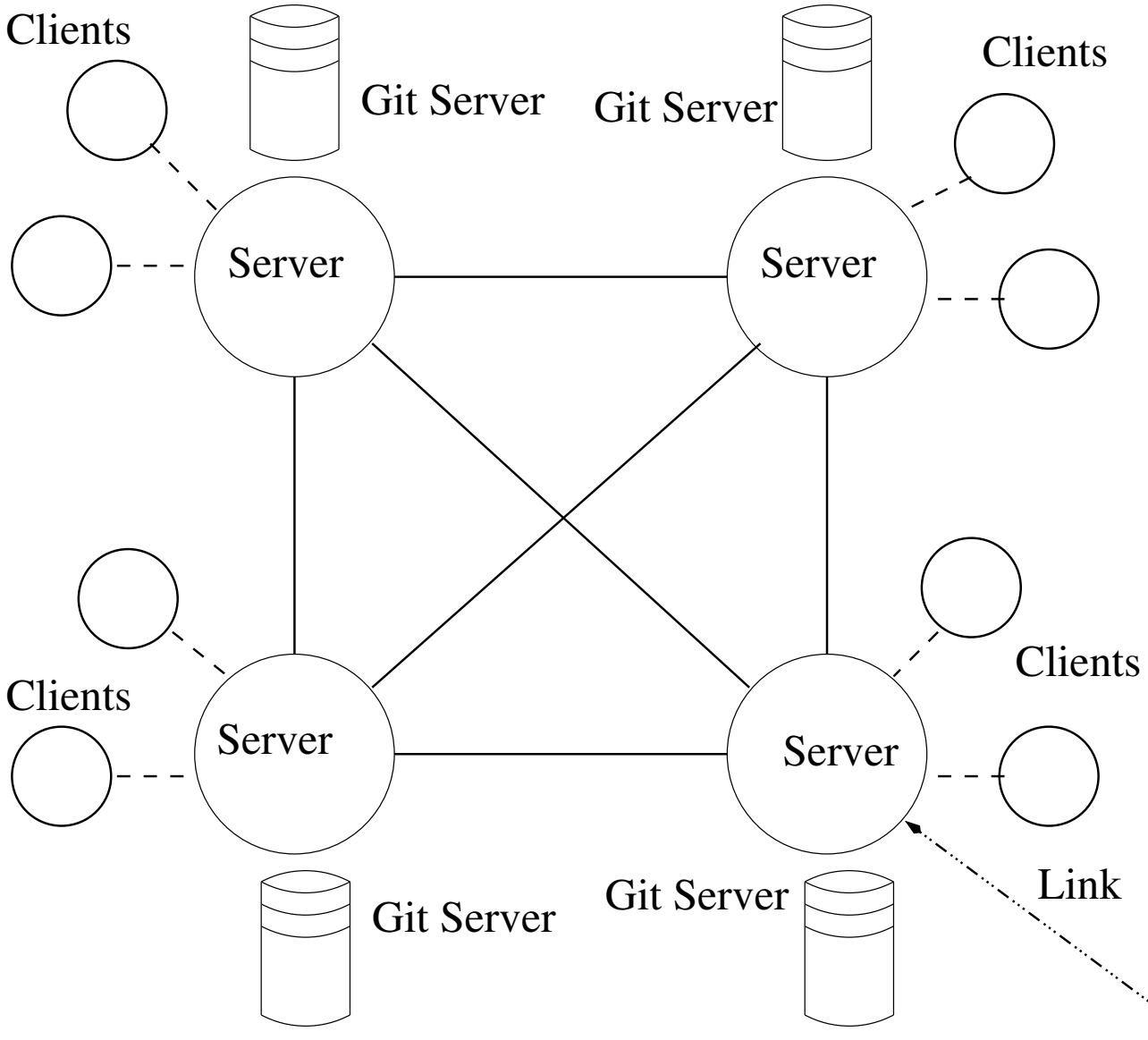
- The certification process checks the results of the reports generated in the verification phase.

```
llr_conforms_to_standards(LLR, Report) :-
    design_llr_conforms_to_standards(LLR, Report),
    ...,
    design_chart_lang_functionrules_conformance(Report, 'Yes'),
    ... .
```

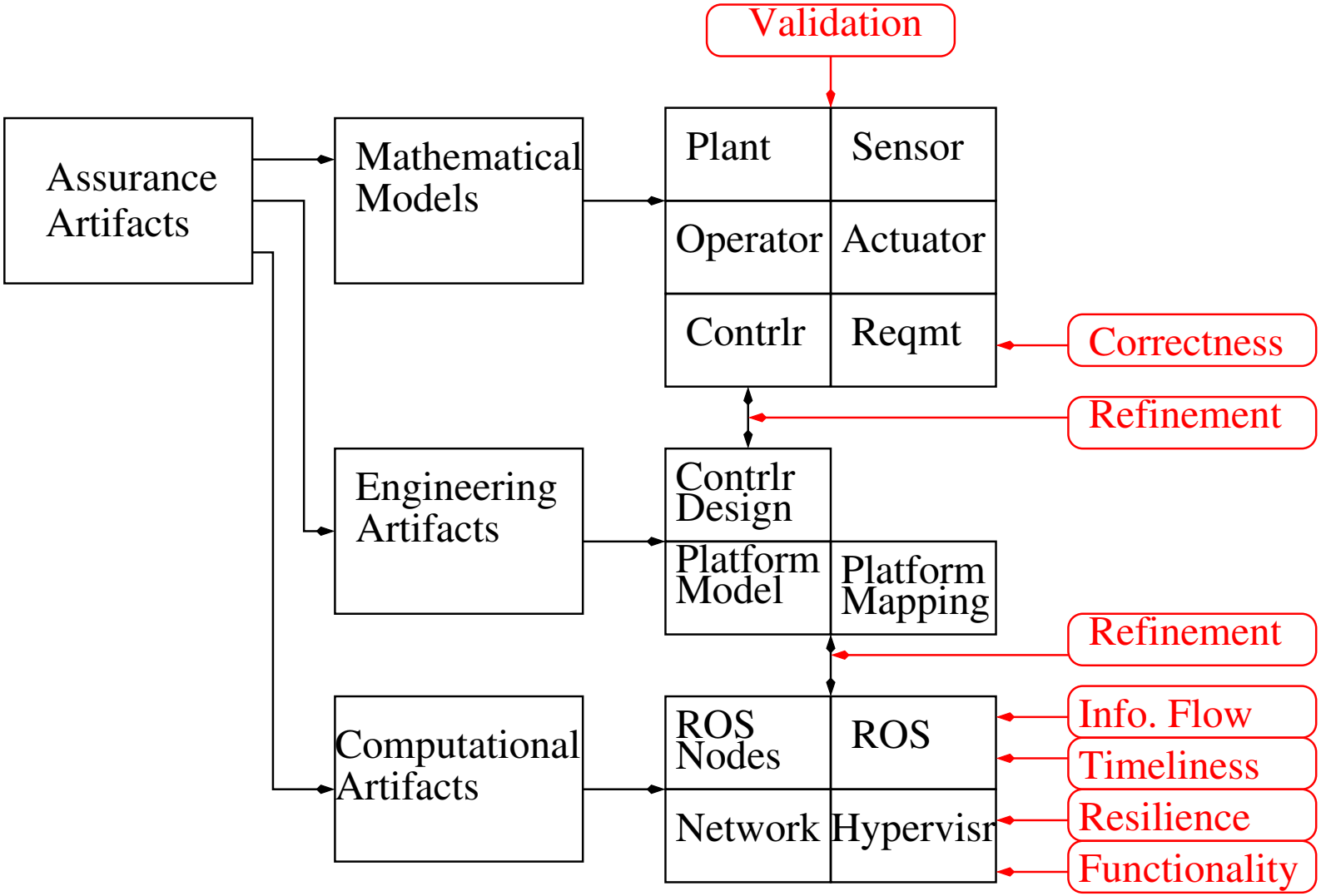
-

```
do178c_a4_check(AA) :-
          a4verify(AA, A4Evidence)
          unpack_evidence(A4Evidence, ...,
                          VerificationReport_5, ...)
          ...
          llr_conforms_to_standards(LLR, VerificationReport_5),
          ... .
```

ETB Architecture



Landshark Assurance in ETB



Conclusions

- Cyber-physical systems range from engine controllers, cars, and robots to factories, buildings, and power grids.
- The incorporation of software and networking makes the safety and security of these systems a critical challenge.
- The construction of the assurance case should drive the design of cyber-physical systems.
- The assurance-driven design (ADD) starts with an eight-variables model of the system developing three layers of design and assurance: the mathematical, engineering, and computation layers.
- The assurance argument and artifacts are assembled using the Evidential Tool Bus (ETB).
- Our approach is currently being applied to the Landshark Robot and will subsequently be adapted to the American Build Automobile.