Building Assurance Cases with the Evidential Tool Bus[1]

N. Shankar

Computer Science Laboratory
SRI International
Menlo Park, CA

Mar 4, 2014

[1]Supported by NASA Cooperative Agreement NNA10DE73C, NSF Grant CSR-EHCS(CPS)-0834810, DARPA. 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- From the FDA Draft Guidance document *Total Product Life Cycle: Infusion Pump - Premarket Notification [510(k)] Submissions*:

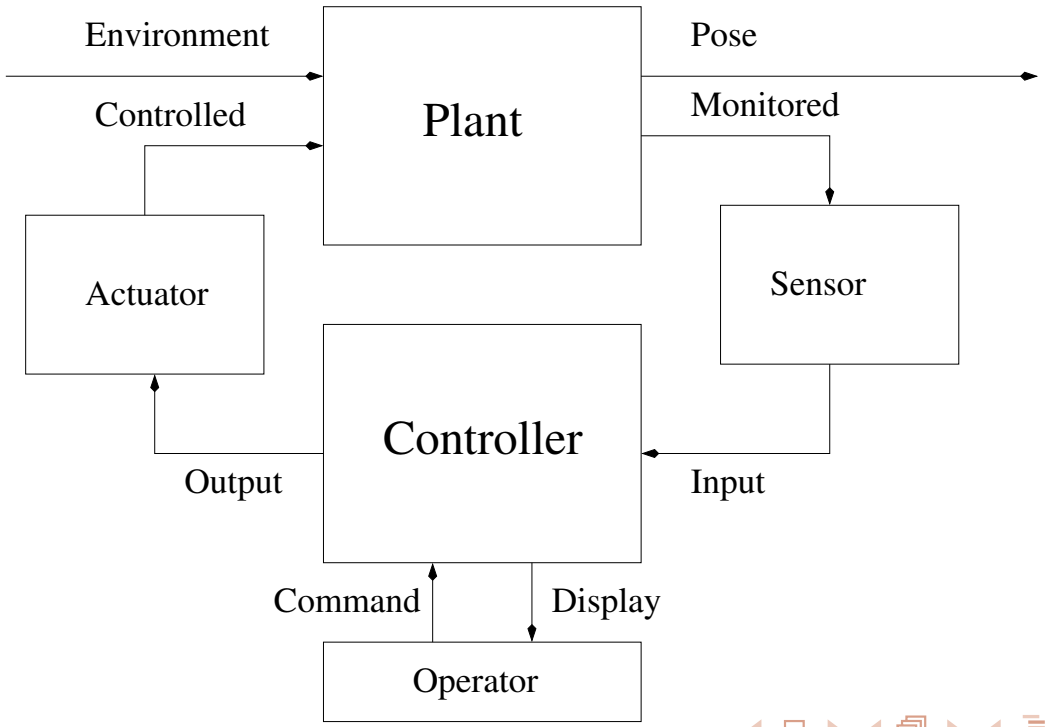
  An assurance case is a formal method for demonstrating the validity of a claim by providing a convincing argument together with supporting evidence. It is a way to structure arguments to help ensure that top-level claims are credible and supported. In an assurance case, many arguments, with their supporting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

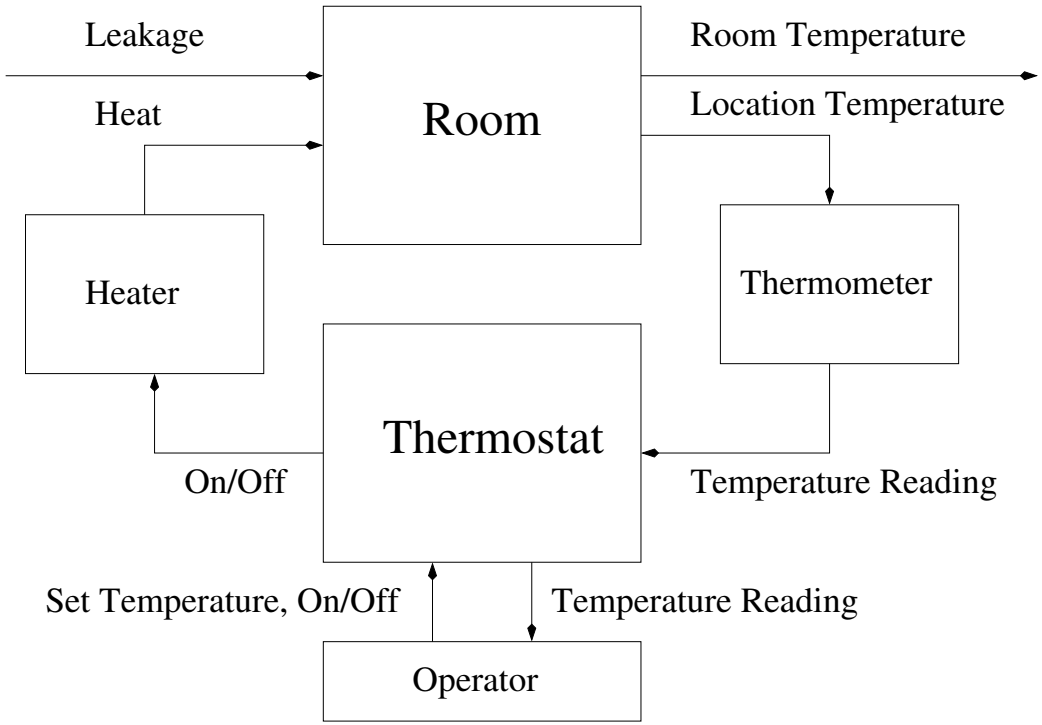
- Key Challenges are
  - How do we systematically construct assurance cases consisting of claims, evidence, and arguments (formal, semi-formal, and informal)?
  - Can we make assurance an integral goal of the design process?
- Assurance for cyber-physical systems
  - Eight-variables model
  - Layered assurance of cyber-physical systems: The Landshark Example
- The Evidential Tool Bus
  - What is the Evidential Tool Bus?
  - How is it used in defining assurance workflows and arguments?
- *Talk describes ongoing work in the HACMS project ground team (SRI, GM/HRL, CMU, MIT, Princeton, Yale, Penn/UCLA).*
- *Please interrupt with questions.*

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.

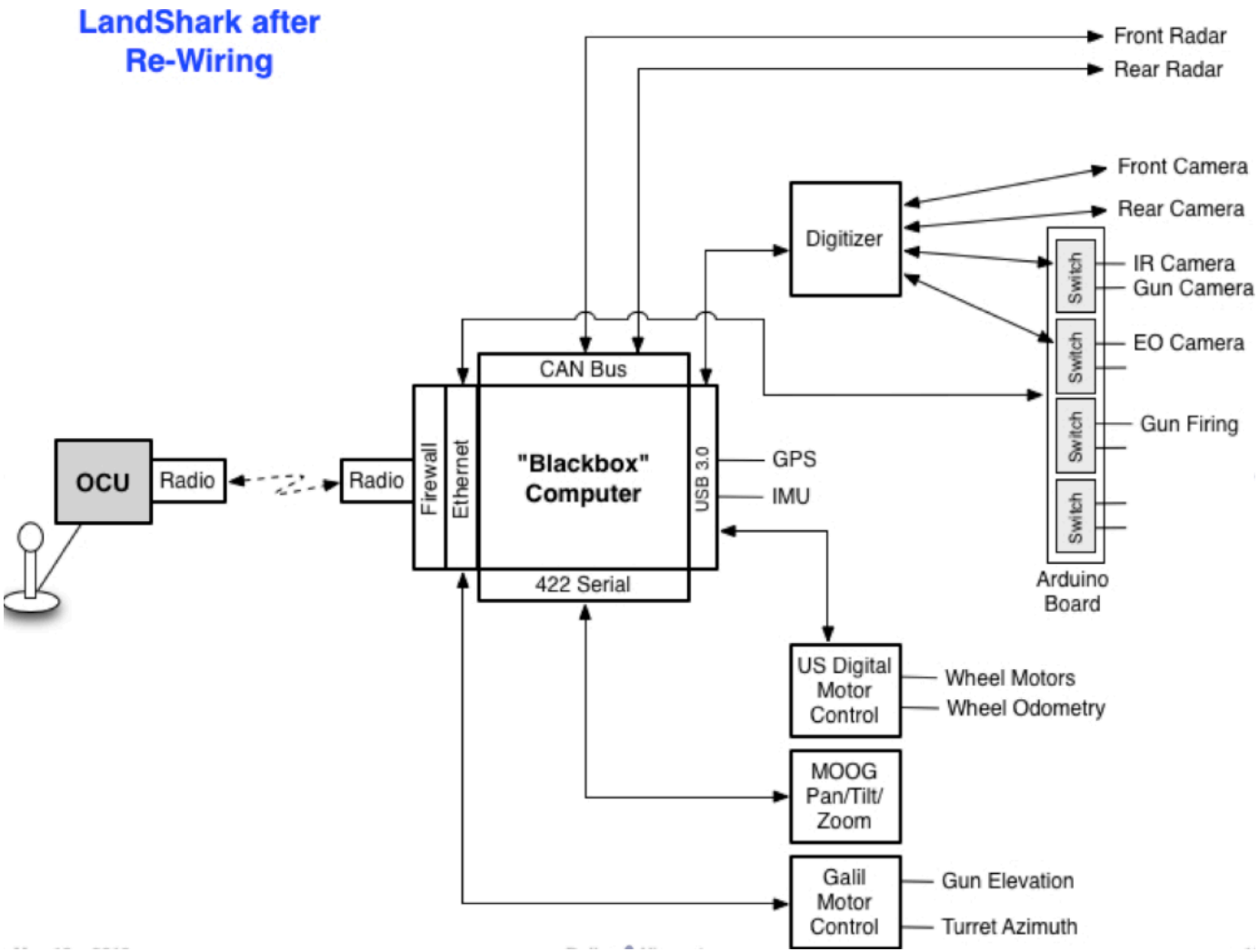


A Simple Example: Room-Heating Thermostat



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.

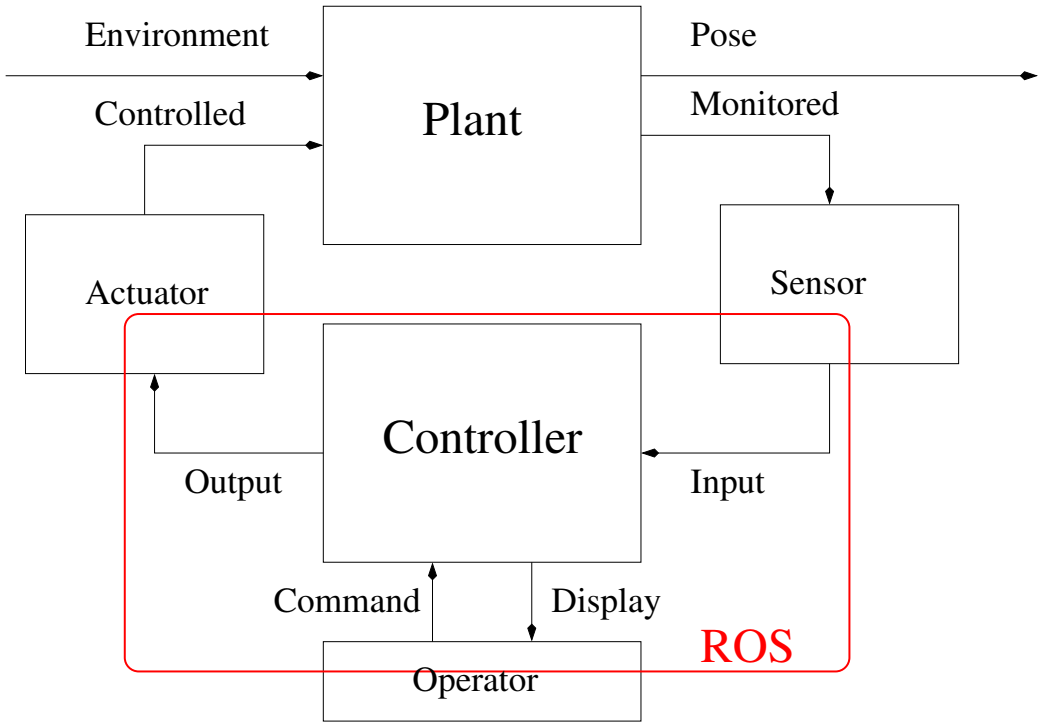
The Landshark Ground Vehicle [Bolles-Vincent]



The vehicle consists of the Operator Console Unit (OCU) and the Unmanned Ground Vehicle (UGV) running the Blackbox.

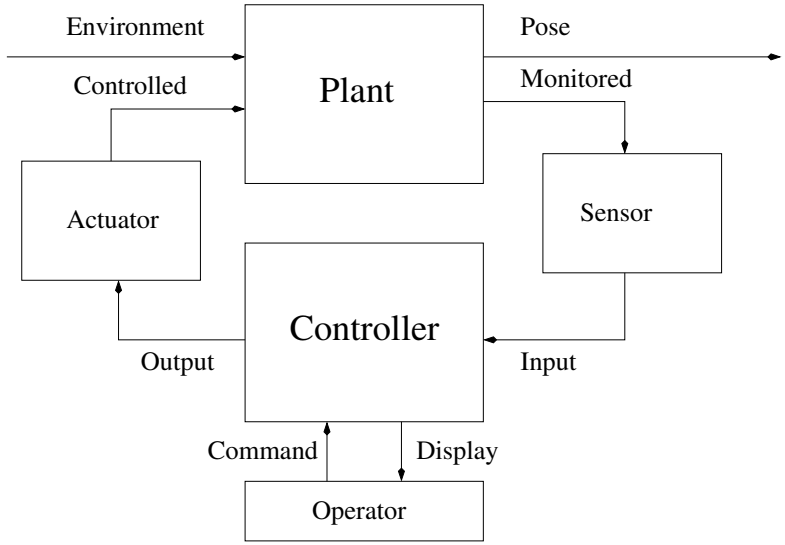
The ROS Platform

The Robot Operating System (ROS) provides a publish/subscribe architecture which is used in the HACMS Landshark.



The software for the sensors, controllers, actuators, and operator/display are in ROS nodes that communicate through ROS messages.

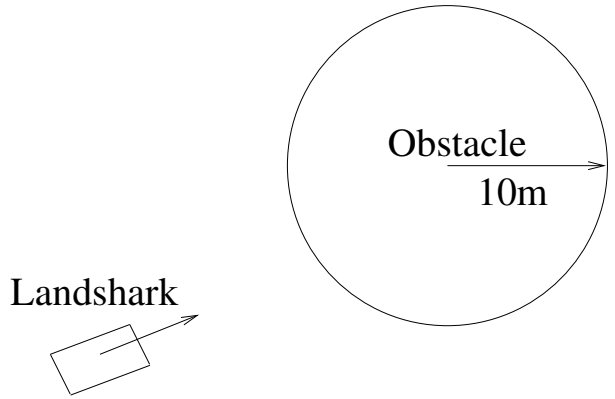
Top Assurance Claim




```
  EnvironmentAssumption(environment) AND
  PlantModel(environment, control, pose) AND
  MonitorPose(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)
```

A Target Requirement: Obstacle Avoidance

- As a challenge property for end-to-end assurance, we target an obstacle avoidance (OA) maneuver.



- The obstacle is given as a GPS point.
- The main requirement is that in responding to OCU inputs, the UGV must avoid bringing the vehicle to within a ten meter radius of the obstacle.
- If the vehicle is already in violation of the obstacle, it only accepts inputs that are guaranteed to move the vehicle away from the obstacle zone.

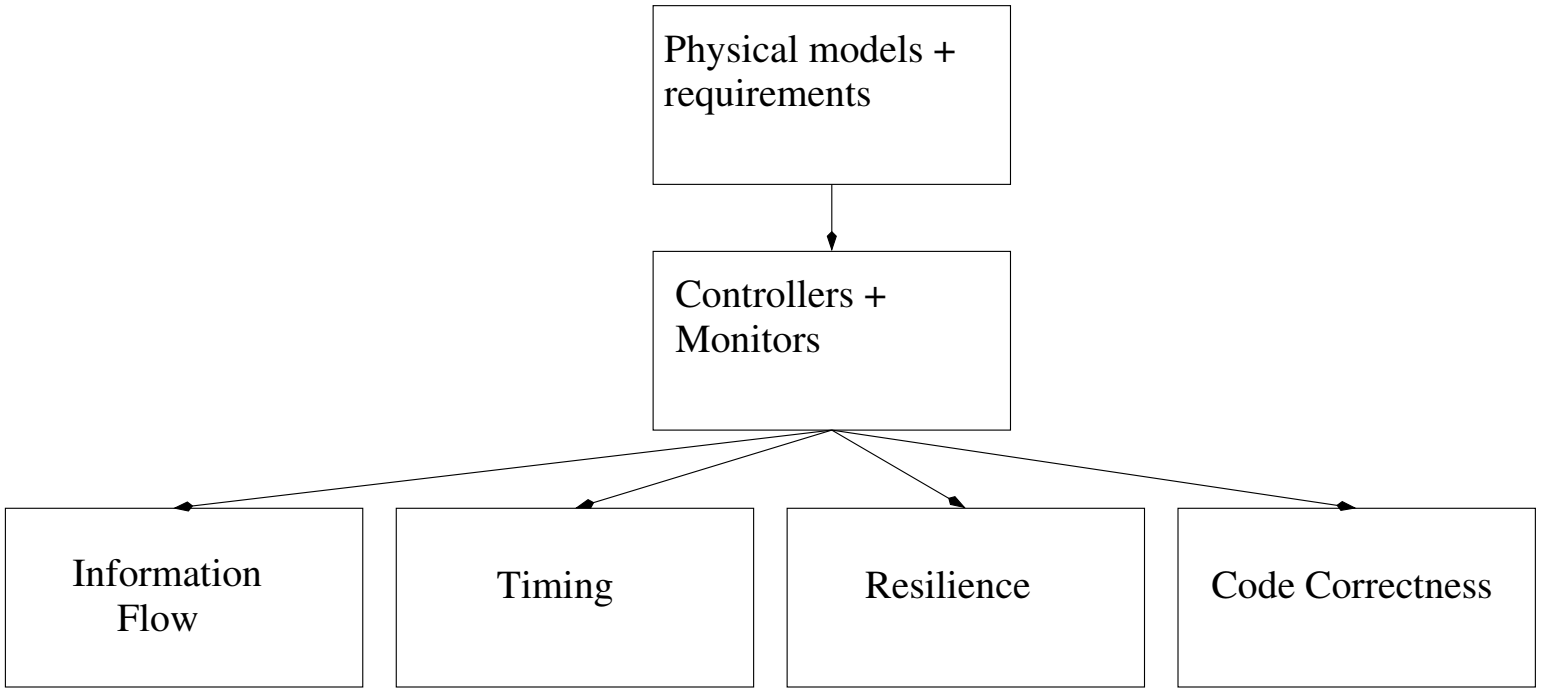
What are the Threats/Hazards?

- The assumption is that the attacker does not have physical access to the vehicle system.
- The attacker can
  1. Physically observe the UGV
  2. Read and record any signal traffic from the OCU or UGV
  3. Insert signals into the traffic going into the vehicle, including GPS signals
  4. Exploit software bugs, sensor failures, and cryptographic vulnerabilities.

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 platform models for communication and computation.
  3. **The Computation Model:** The software for modules in the engineering model are turned into ROS nodes executing 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.

A Multi-Legged Argument



The actual platform must be functionally correct, resilient, timely on both the communication and computation, and preserve data integrity and authenticity.

The Mathematical Argument

- The high-level requirement requires the vehicle to follow the operator commands without entering the obstacle zone.
  - A state estimation algorithm captures an accurate estimate of the pose of the vehicle as long as a majority of sensors are reliable to within a given bound. [Penn]
  - A obstacle avoidance controller blocks commands from the OCU that, based on the state estimate, the actuator response, and vehicle dynamics, might cause the vehicle to violate the obstacle zone. [CMU]
  - The OCU operator commands are delivered to the controller in a timely manner

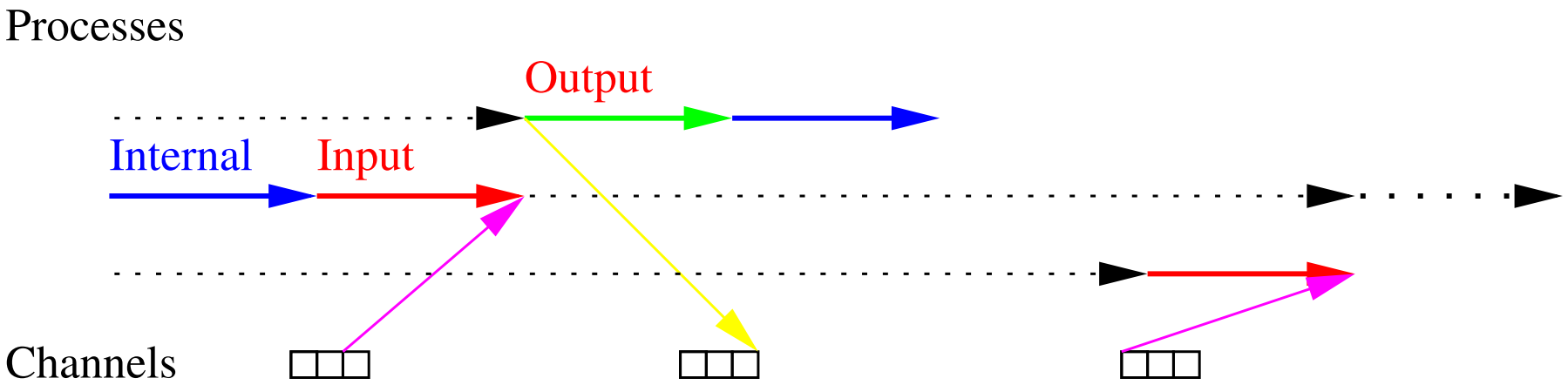
The Engineering Model

- The engineering model introduces two controller components:
  1. A Reliable State Estimator (RSE) that computes an estimate of the position and velocity of the vehicle while allowing for a minority of faulty sensors.
  2. An Obstacle Avoidance Controller (OAC) that uses the state estimate from the RSE to ignore or respond to incoming commands
- The platform model assumes a timed synchronous model of computation with
  1. Periodically scheduled processes
  2. Communicating through a timely and reliable (integrity + authentication) channels

Computational Model

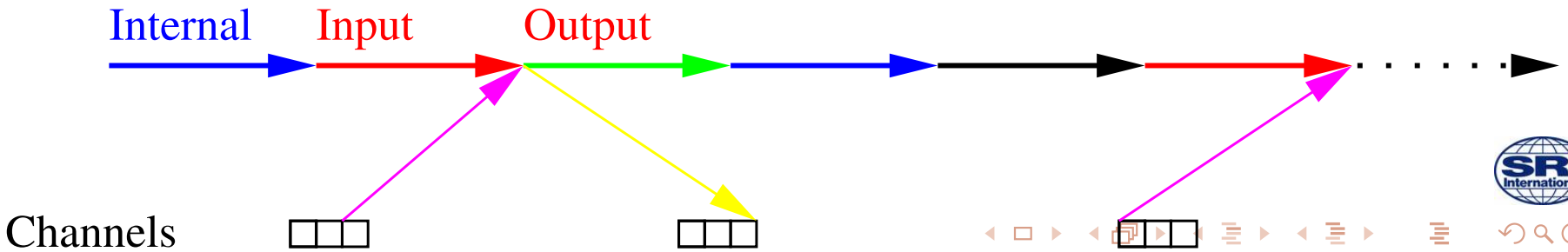
- The computational model uses the Robot Operating System (ROS) platform with
  1. A start-up procedure that places the Landshark system in a stable state where the partitions, nodes, devices, communication links have been initialized.
  2. A scheduler that coordinates the execution of the tasks to satisfy the platform assumptions of the engineering model
  3. Authenticated communication between ROS nodes
- The ROS nodes on the vehicle run in separate partitions of the CertiKOS hypervisor using inter-partition communication.
- All message traffic from external nodes are filtered through a firewall (to ensure that no internal addresses are spoofed).

Low-level Assurance: MILS-based Information Flow



Processes can perform reads (from allowed channels), writes (to allowed channels), and internal actions, and modify local state.

The projection of a computation on the state and channels for a single process is a valid computation for the single process.



Low-level Assurance: Resilience

- The claims ensure that the obstacle avoidance requirement holds even when the system is attacked.
  1. Resilience to sensor faults/infiltration is ensured by RSE correctness
  2. Resilience to attacker infiltration (including through replay attacks) is ensured by the information flow argument
  3. Resilience to timing uncertainty is ensured by timeliness of computation and message delivery (with strong assumptions).
  4. Resilience to node failure: untrusted nodes are separated from trusted ones by the hypervisor.

Low-level Assurance: Code Correctness

- Trusted code base includes
  1. Message signing and authentication
  2. ROS Master configuration node
  3. Runtime Verifier node for ROS messages
  4. CertiKOS hypervisor
  5. Network stack
  6. ROS libraries
  7. ROS nodes such as RSE and OAC
- A few of these such as message signing/authentication, ROS master, runtime verifier, and some network capabilities, are being formally verified.

Construction of Assurance Case

- Definition of model layers: PVS
- Model properties: HybridSAL, KeyMaera
- Architecture/Platform definition: Simulink, SAL, ROS
- Code correctness: Bedrock, Spiral, K, VST
- CertiKOS: Coq
- Authentication: Coq
- Network stack: Specware
- Semantic integration of claims: PVS
- Syntactic integration of assurance artifacts and claims: ETB

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.

Why a Tool Bus?

- Verification technology has reached an impressive level of maturity.
- But most formal analyses, and software design workflows, are built from a combination of front-end and back-end tools, e.g.,
  - Counterexample-guided abstraction refinement (CEGAR) using model checking and predicate abstraction
  - Requirements validation, concolic test generation, and code coverage evaluation
  - Assertional verification combining static analyzers and decision procedures
  - Model-based design tools supporting simulation, verification, code generation, and runtime monitoring
- *We need a framework for defining workflows that integrate multiple tools to produce verifiable arguments chaining claims and evidence.*

Why a *Distributed* Evidential Tool Bus?

- Many applications of ETB require a distributed implementation: e.g., distributed `make`, regression testing.
- We also want to have dedicated tool servers that can be invoked as services.
- Many tool services only run on specific platforms, e.g., Linux, Windows.
- Assurance cases are often constructed in a distributed manner.
- The ETB architecture lends itself naturally to a distributed implementation.

What ETB is Not?

- *Fine-grained integration* (Shostak and Nelson–Oppen theory combinations): *This would incur a high overhead if implemented with ETB.*
- *Semantic interchange frameworks* (PROSPER, ToolBus, Veritech, Boogie, Why, and Frama-C): *ETB makes no semantic commitments.*
- *Metalogical frameworks* (LF, Isabelle, and Twelf): *ETB's metalogic is simpler and less sophisticated.*
- *Web service architectures and languages* (ETI, Orc): *These are not "evidential".*
- *Middleware for coarse-grained distributed computing* (Condor and Hadoop): *These focus is on high-performance, not evidence tracking.*
- *ETB is a semantically neutral, evidence-based distributed platform for heterogeneous tool integration.*

ETB Desiderata

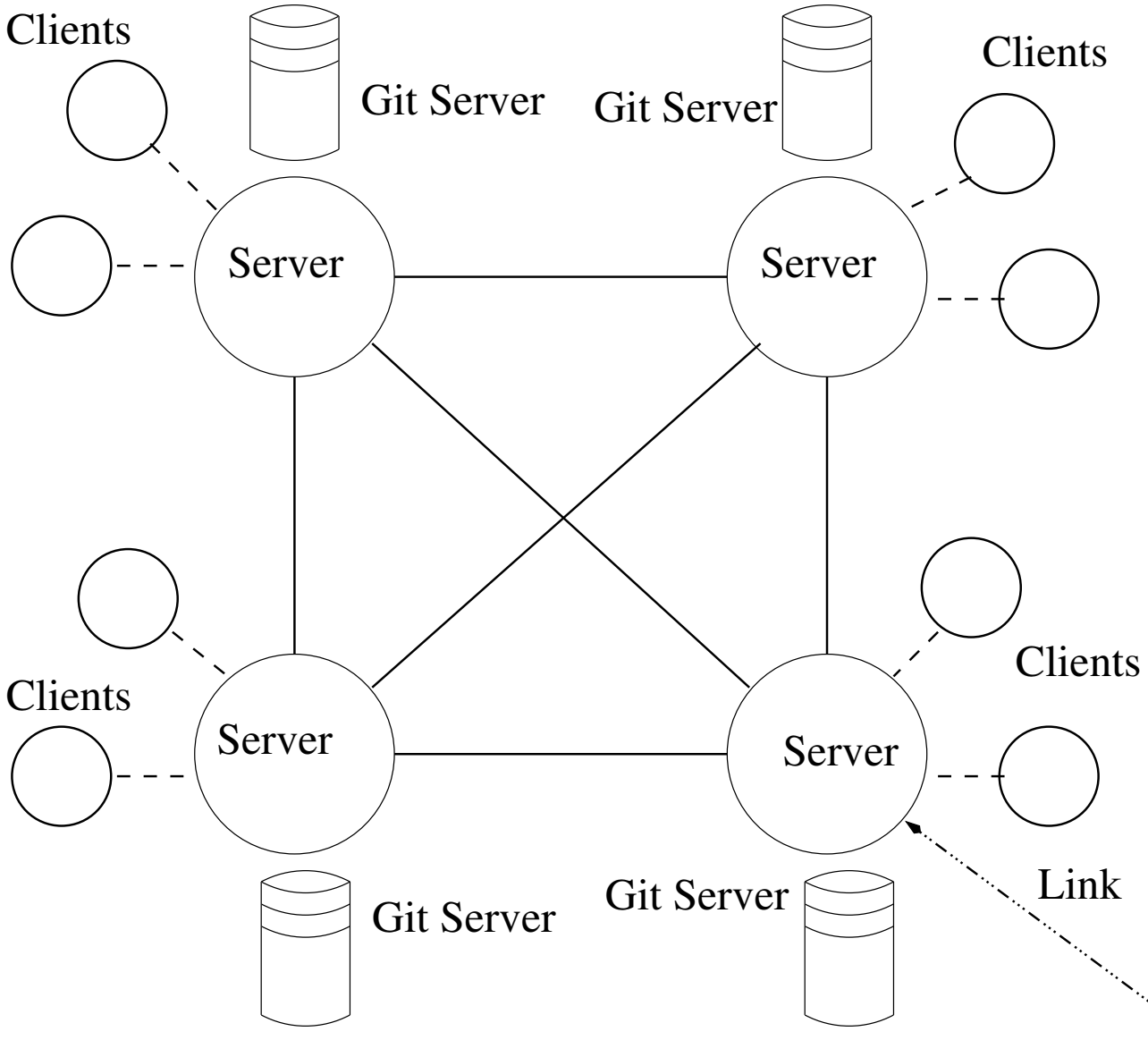
- ETB targets the production of claims supported by arguments in which some of the sub-claims can be established by external tools.
- The three key design requirements for ETB are

  Extensibility:
  - New claim forms and rules of argumentation
  - New external tools (including human oracles)
  - New workflows that are defined by scripts
  - New clients and servers

  Assurance:
  - Explication of tools, artifacts, rules, and assumptions on which the argument depends
  - Replay and rechecking of arguments

  Semantic Neutrality: ETB makes no commitments to specific tools, languages, or models
- ETB is infrastructure for building and checking arguments, and can be used to implement specific assurance methodologies.

ETB Architecture



ETB Implementation: Servers

- An ETB cluster is a fully connected network of servers.
- Servers can join and leave an ETB cluster, and *link* to other ETB networks (e.g., through SSH tunneling).
- Each servers includes
  - A *claims table* that records all the valid claims established by or subscribed to by the server.
  - A *git repository* as a workspace — file handles are `git`'s SHA-1 hash.
  - A *logic engine* for evaluating queries.
- Servers can advertise (*publish*) services and *subscribe* to claims through a shared heartbeat.
- Servers can run on different machine configurations as needed to support individual tools, e.g., Linux or Windows.

The ETB Stack

- The ETB stack consists of the following layers
  1. **Network:** Maintains the connectivity status of the ETB cluster
  2. **Service:** Servers can publish services and subscribe to claims
  3. **File:** Each server operates a `git` repository and files/directories are copied by other servers as needed.
  4. **Session:** Servers exchange claims and queries, as needed to invoke external tools
- ETB also provides support for visualization and error handling for distributed computations over an ETB network,

ETB Clients

- ETB offers an XML-RPC API for clients.
- Currently, we have client APIs for Python, C, Java, OCaml, and an ETB shell.
- The API can be used to
  - Connect a new ETB node
  - Import a file into ETB and generate a handle
  - Copy a file to the local `git` from the ETB network
  - Launch a query
  - Get the completed answers when a query terminates, or partial answers
  - Get the claims generated by a query
  - Visualize the evaluation of a query

Datalog as a Metalanguage

- *Atoms* are of the form $p(a_1, \ldots, a_n)$, where p is a *predicate* and each a_i is either a *data object* or a variable, e.g.,
  - *models*(*Model*, *Formula*)
  - *satisfiable*(*Formula*)
  - *cnf*(*Formula*, *CNFFormula*)
- A *predicate* p can either be
  - *Interpreted* by means of a tool invocation through wrappers, e.g., `yices` which invokes the Yices SMT solver. Both the tool and wrapper have to be trusted.
  - *Uninterpreted*, i.e., defined by a Datalog program that is evaluated locally, e.g., `allsat` which we define to use Yices to compute the DNF equivalent of a given formula.
- A *query* is an atom (possibly) with free variables, e.g., *cnf*(*formula*, *CF*).
- A *claim* is a *ground* atom that is supported by a proof.

The Data in ETB's Datalog

- Data objects are represented as JSON terms.
- The data objects include
  - *Primitive JSON terms* for numbers, strings, objects, and arrays.
  - *File handles*: File name + SHA-1 hash
  - *Tool handles*: E.g., BDD handles, PVS theories
  - *Session handles*: E.g., Yices interactive contexts
- *Claims that hold of earlier file version need to be recomputed for any new versions.*

An Example ETB Workflow: AllSAT

The defined predicates `sat` and `unsat` invoke the interpreted `yices` predicate on the given file `F`.

```
sat(F, M) :- yices(F, S, M),
             equal(S, 'sat').
unsat(F) :- yices(F, S, M),
            equal(S, 'unsat').

allsat(F, Answers) :- sat(F, M),
                      negateModel(F, M, NewF),
                      allsat(NewF, T),
                      cons(M, T, Answers).
allsat(F, Answers) :- unsat(F),
                      nil(Answers).
```

Though `allsat` calls `sat` and `unsat`, the `yices` wrapper is only executed once on the file `F` since the resulting claim is tabled.

A Variant: AllSAT with a Yices Session

```
allsat(F, Answers) :- yicesStart(Session0),
                      yicesIncludeFile(Session0, F, Session1),
                      allsat_enum(Session1, Answers).

allsat_enum(Session, Answers) :-
      yicesCheck(Session, Session1, Result),
      allsat_iter(Session1, Result, Answers).

allsat_iter(Session, Result, Answers) :-
      equal(Result, 'sat'),
      yicesModel(Session, Model),
      yicesAssertNegation(Session, Model, Session1),
      allsat_enum(Session1, Answers1),
      cons(Model, Answers1, Answers).

allsat_iter(Session, Result, Answers) :-
      equal(Result, 'unsat'),
      yicesClose(Session),
      nil(Answers).
```

ETB Datalog versus Datalog

- Datalog as a database query language has intensional and extensional predicates — ETB has uninterpreted and interpreted predicates.
- Interpreted predicates are similar to built-ins (which evaluate ground atoms), but more general.
- ETB only admits top-down left-to-right evaluation — no bottom-up evaluation.
- In ETB, Datalog is the metalanguage — sparse data, but elaborate workflows.
- In Datalog, the Herbrand universe is bounded, but in ETB Datalog, it is unbounded.
- ETB Datalog evaluation is distributed — uninterpreted predicates are evaluated locally, and interpreted predicates might be remotely evaluated.
- No (stratified) negation — we only establish positive claims.

What is an Interpreted Predicate?

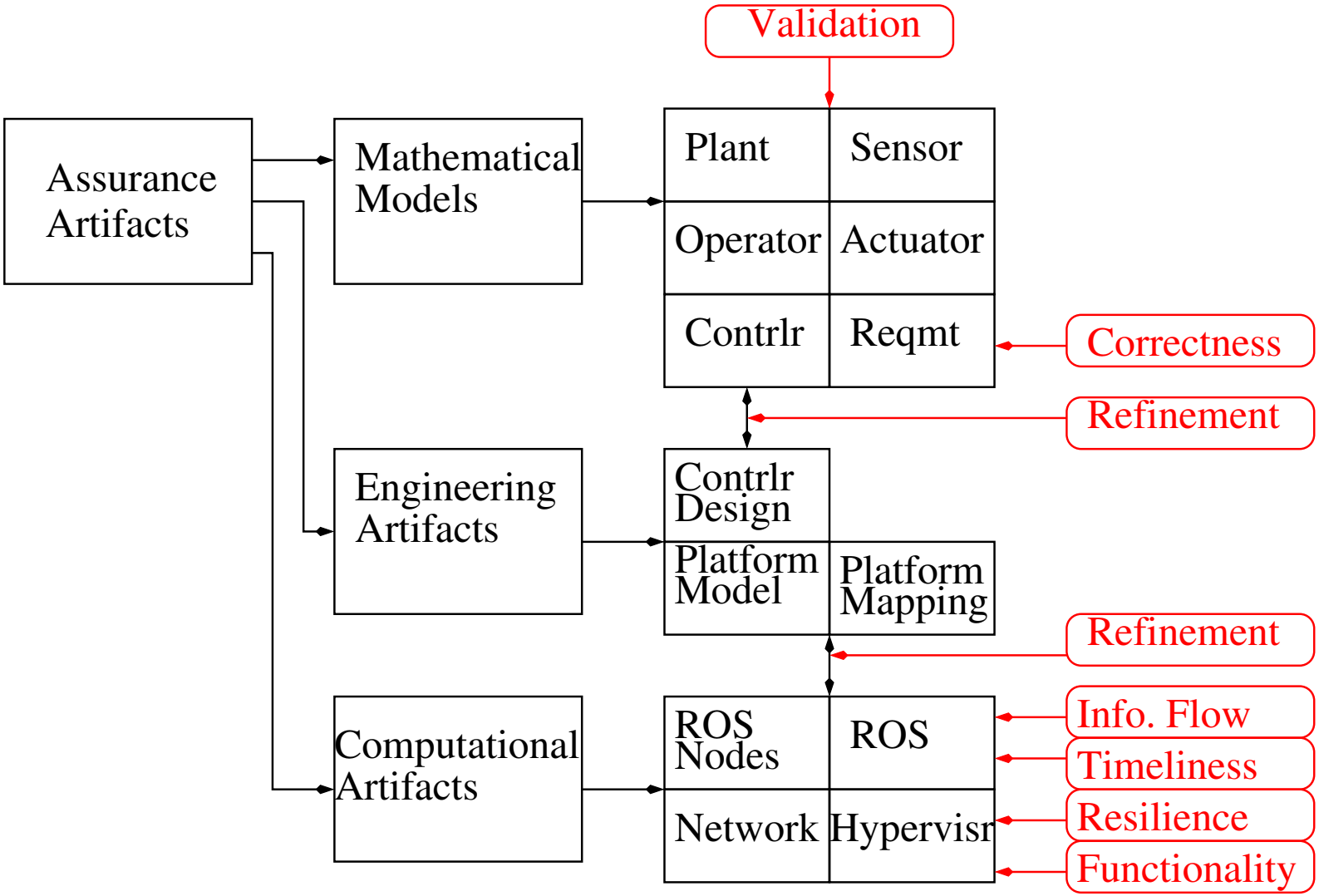
- Datalog has been extended with built-in predicates, but these are usually evaluated when all arguments are grounded, e.g., $<(x, y)$.
- An interpreted predicate $p(a_1, \ldots, a_n)$ is evaluated by a wrapper.
- The evaluation of a predicate $p(a_1, \ldots, a_n)$ generates clauses (lemmata) of the form

$$\begin{array}{rcl} p(b_{11}, \ldots, b_{1n}) & :- & Q_1 \\ & \vdots & \\ p(b_{m1}, \ldots, b_{mn}) & :- & Q_m \end{array}$$

- For example, the evaluation of $\texttt{veryComposite}(8, 3)$, can generate

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

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.