

Leverageable Semantics Definitions and Contract Reasoning for a Technical Architecture Description Language

HCSS 2023 – May 8, 2023

Kansas State University

*Software Engineering Institute
(Carnegie Mellon University)*

Galois

John Hatcliff

Jason Belt

Robby

Jerome Hugues

Lutz Wrage

Danielle Stewart

Todd Carpenter

Ryan Peroutka

August Schwerdfeger

+ collaborators from Collins Aerospace and sel4 team

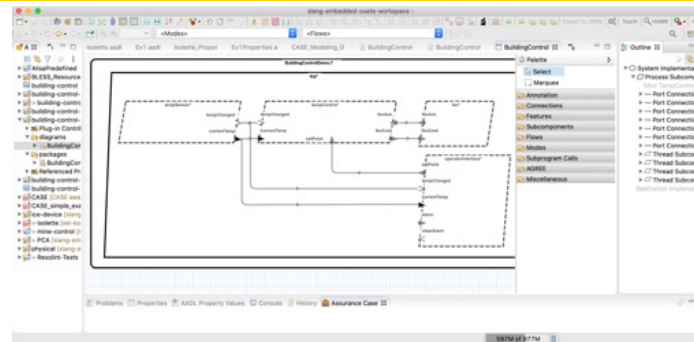
This material is based on research sponsored in part by US Army and DARPA

DISCLAIMER: The views and conclusions contained in this presentation are those of the author and should not be interpreted as representing the official policies, either express or implied, of any agency or department of the U.S. Government, Kansas State University or Galois, inc

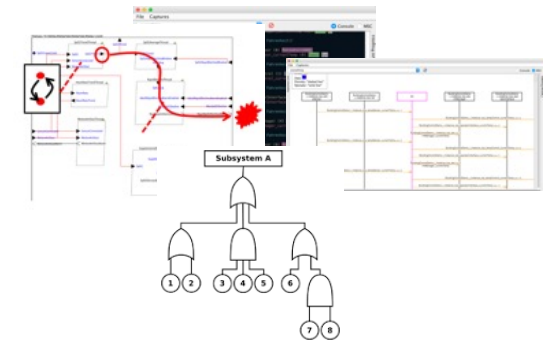
HAMR

HAMR – tool chain for model-driven development of high-assurance embedded systems
(from Adventium Labs/Galois and Kansas State)

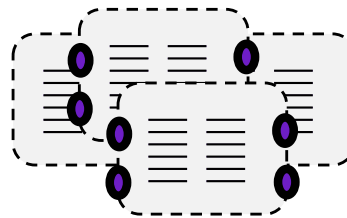
Modeling, analysis, and verification in the **AADL** modeling language



Leveraging analyses from AADL community



Component development and verification in multiple languages



- C
- Slang
 - high integrity subset of Scala
 - contract verification framework
 - translates to C
- CakeML (ML-variant with verified compiler)

Deployments aligned with AADL run-time on multiple platforms



Slang Contracts and Automated Verification via Symbolic Execution (Logika)

Slang – high-integrity subset of Scala + Logika verification in IntelliJ IDE

The screenshot shows the IntelliJ IDE interface with a Slang file named `sp_queue_int8_t_dropOldest-sp-verified.sc`. The code is divided into two main sections:

- Slang Contract (Lines 90-101):** A contract for the `dequeue` function. It includes preconditions (`Requires(numOfElements > 0, dataOut.size == 1)`), a postcondition (`Ensures(numOfElements == In(numOfElements) - 1, ...)`), and a `Contract` block with various assertions about the state of the queue and the output.
- Application Code (Lines 103-116):** The implementation of the `dequeue` function. It checks if the queue is empty, gets the number of elements, and then dequeues an element from the queue, updating the state and the output.

On the right side, the **Sireum** output console shows the results of the verification:

- Validity Check for Implicit Indexing Assertion at [111, 22]: Valid**
- Validity Check for Implicit Indexing Assertion at [111, 11]: Valid**
- Result: Valid**
- Solver: /Users/robby/Repositories/Sireum/kekinian/t**
- Arguments: --tlimit=2000 --lang=smt2.6 --full-saturate**
- Sequent:** A complex logical expression representing the verification goal.

Verification Drill-down Controls

Slang applications can be integrated with Scala and Java and executed on JVM or transpiled to JS or C. The generated C has bounded memory usage and no garbage collection & compatible with verified CompCert compiler.

Logika Verification

Featureful, Integrated Capabilities

Logika uses a **server-based architecture** with a suite of SMT solvers (Z3, CVCx, Alt-Ergo), massive parallelization, with “always on” smart incremental checking

Logika verification of Slang code in IntelliJ IDE on iPad

```
def perform_fan_control(api: TempControl_i_Operational_Api) : Unit = {  
  Contract(  
    Requires(api.fanCmd == None[FanCmd.Type]), // for now we need to ma  
    Modifies(api, // modifies api.fanCmd  
             currentFanState),  
    Ensures(// current have to "manually" give frame-condition for all inp  
            api.currentTemp == In(api).currentTemp,  
            api.setPoint == In(api).setPoint,  
            api.fanAck == In(api).fanAck,  
            // post-conditions  
            (latestTemp.degrees < currentSetPoint.low.degrees) →:  
              (currentFanState == FanCmd.Off),  
            (latestTemp.degrees > currentSetPoint.high.degrees) →:  
              (currentFanState == FanCmd.On),  
            (latestTemp.degrees ≥ currentSetPoint.low.degrees & latestTem  
              →: (currentFanState == In(currentFanState) & api.fanCmd ==  
                (currentFanState ≠ In(currentFanState)) →: (api.fanCmd == So  
            )  
  )  
  
  val oldFanState = currentFanState  
  if (latestTemp.degrees < currentSetPoint.low.degrees) {  
    // if current temp is below low set point,  
    currentFanState = FanCmd.Off  
    api.logInfo("Set fan command: Off")  
  } else if (latestTemp.degrees > currentSetPoint.high.degrees) {  
    // if current temp exceeds high set point,  
    currentFanState = FanCmd.On  
    api.logInfo("Set fan command: On")  
  } else {  
    api.logInfo("Fan state unchanged")  
    // DEMO: Uncommenting this line will lead Logika to find a post-condi  
    // i.e., a fan  
    // api.put_fanCmd  
  }  
}
```

...connected to 80-core server to run verification

From a TCCOE conference demo video of Logika in January 2022

See <https://drive.google.com/uc?export=download&id=1vkBNW8pocSz8jUG-E16zdVleELZr2Sk> for Slang / Logika overview talk given at the Trusted Computing Center of Excellence Symposium

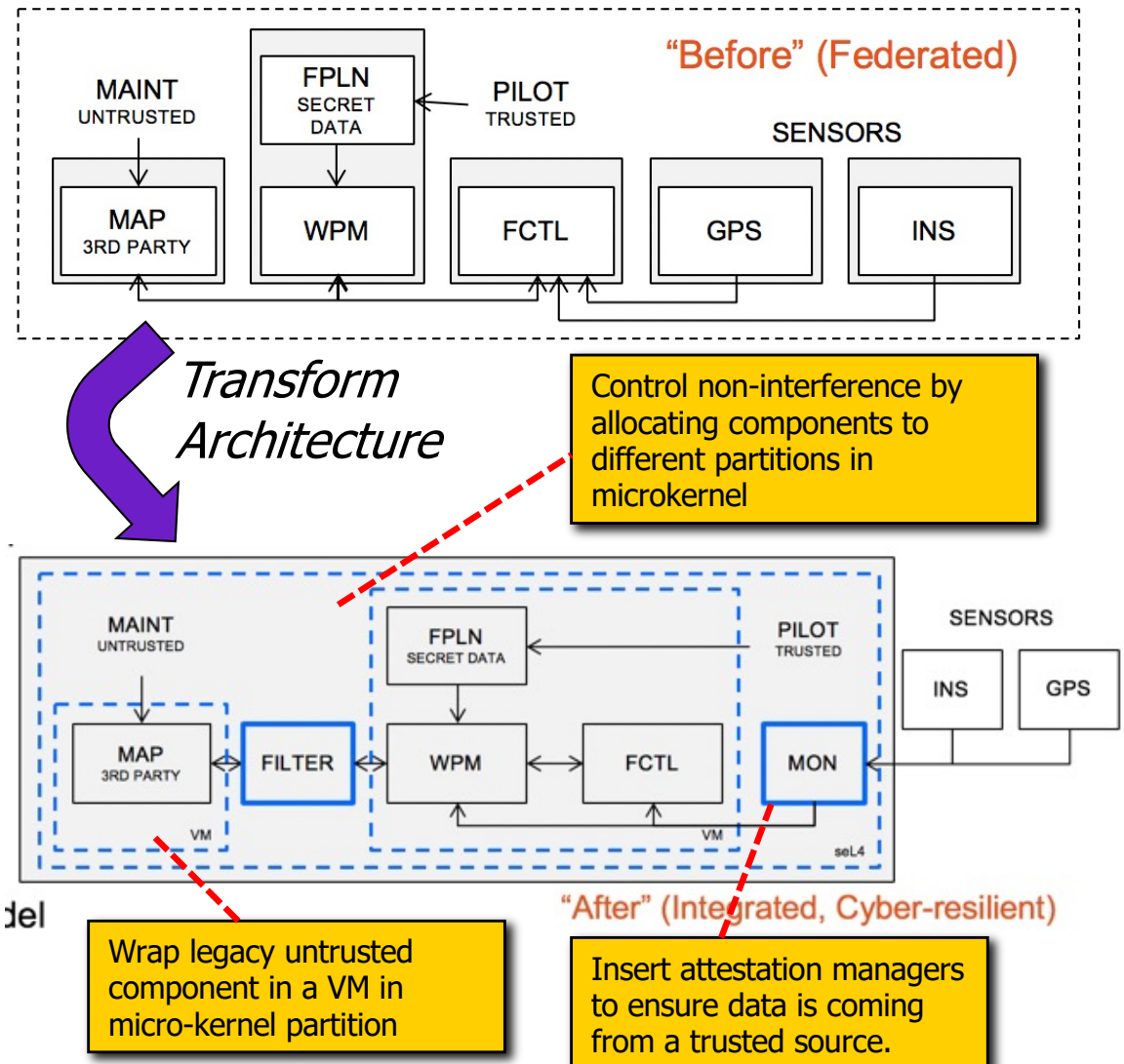
DARPA CASE Approach

HAMR was developed by Kansas State and Galois researchers on a team led by Collins Aerospace (Darren Cofer)

- **Capture requirements** for cyber-resiliency
- **Analyze** design
- **Transform** design
- **Verify new design** against requirements
- **Build / Deploy**

HAMR
Focus

seL4 verified micro-kernel technology is a core technology



DARPA CASE Final Demonstration

HAMR used with Collins BriefCASE tool chain adding new functionality to CH-47 mission computing...

COLLINS COMMON AVIONICS ARCHITECTURE SYSTEM (CAAS)

- Use BriefCASE tools to add new functionality to helicopter avionics system with cybersecurity guarantees
- Demonstrated on CH-47 mission computing platform – relevant for Future Vertical Lift

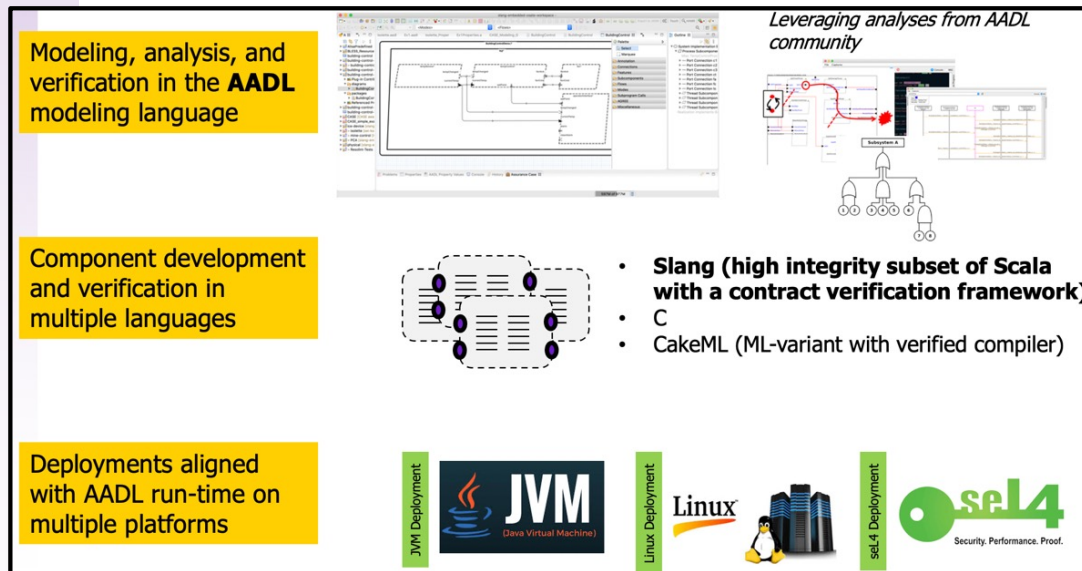


/Released



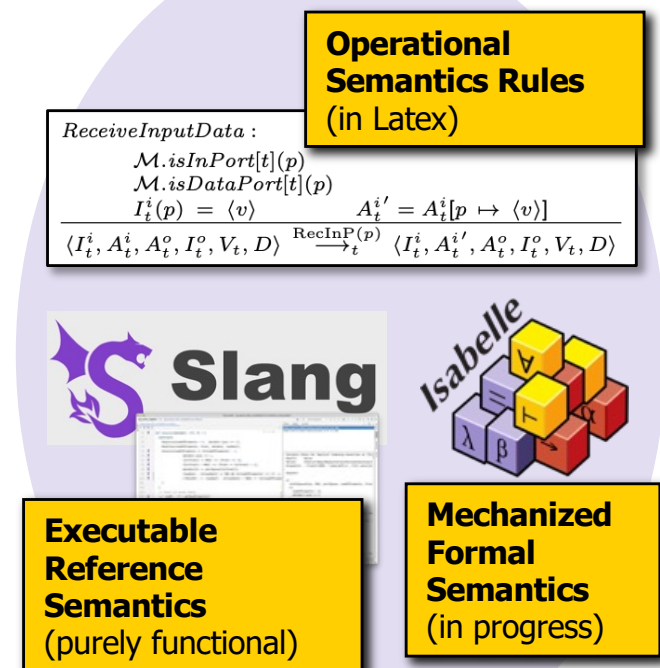
Goal: Semantic Consistency End-to-End

HAMR is supported by a suite of inter-related formal semantics artifacts to aid end-to-end semantic consistency



Analysis and verification results moved up and down abstraction layers

Semantic Consistency

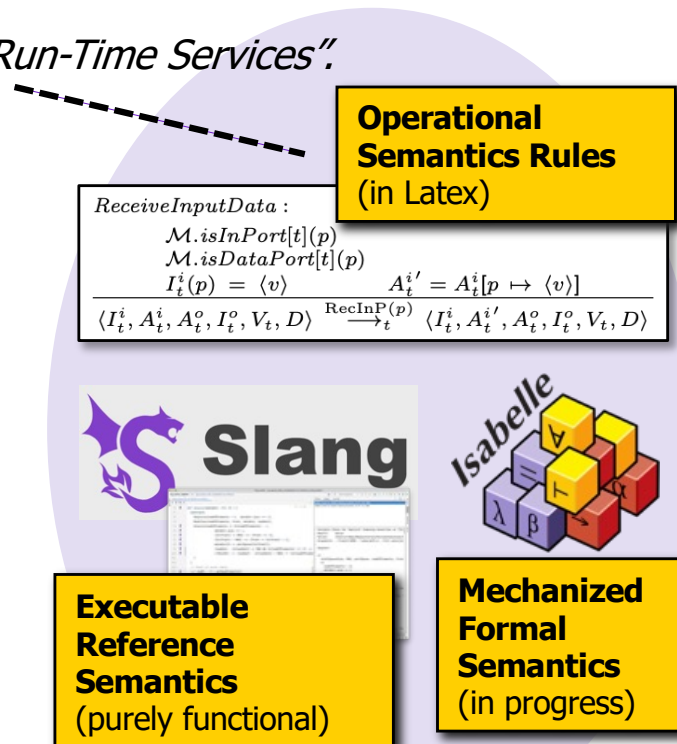


Goal: Semantic Consistency End-to-End

HAMR is supported by a suite of interlated formal semantics artifacts to aid end-to-end semantic consistency

John Hatcliff, Jerome Hugues, Danielle Stewart,
and Lutz Wrage.

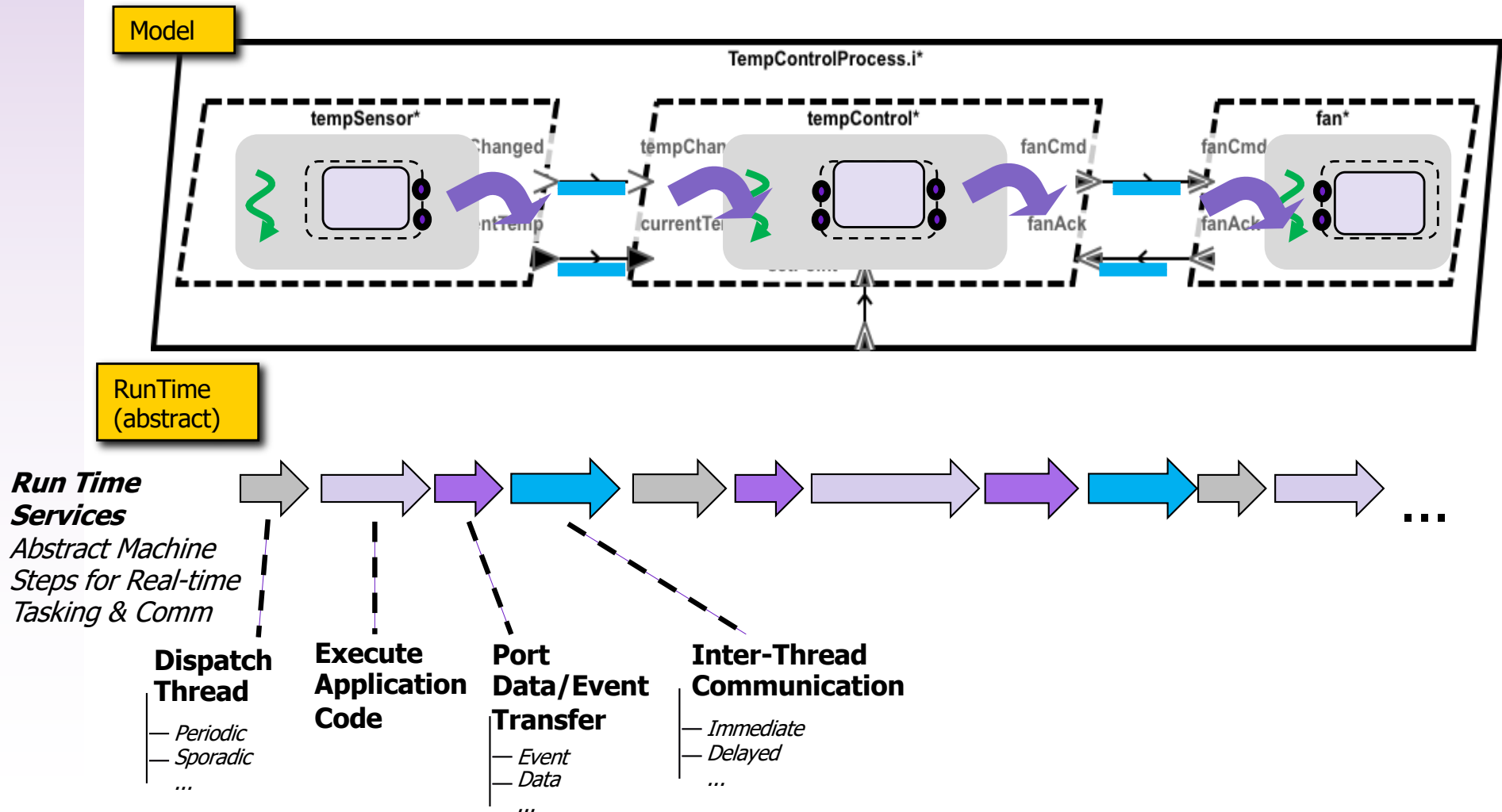
"Formalization of the AADL Run-Time Services".
(ISOLA 2022)



(Kansas State team, University of Aarhus)

Formalizing AADL Run-Time Services

The formal semantics in this talk focuses on the AADL Run-Time Services...



AADL Standard Description is Informal

Description (excerpts) of the **Receive Input** service →
in the previous version of AADL standard...

Narrative description..

A **Receive_Input** runtime service allows the source text of a thread to explicitly request port input on its incoming ports to be frozen and made accessible through the port variables. Any previous content of the port variable is overwritten, i.e., any previous queue content not processed by **Next_Value** calls is discarded. The **Receive_Input** service takes a parameter that specifies for which ports the input is frozen. Newly arriving data may be queued, but does not affect the input that thread has access to (see Section 9.1). **Receive_Input** is a non-blocking service.

```
subprogram Receive_Input
```

```
features
```

```
  InputPorts: in parameter <implementation-dependent>
```

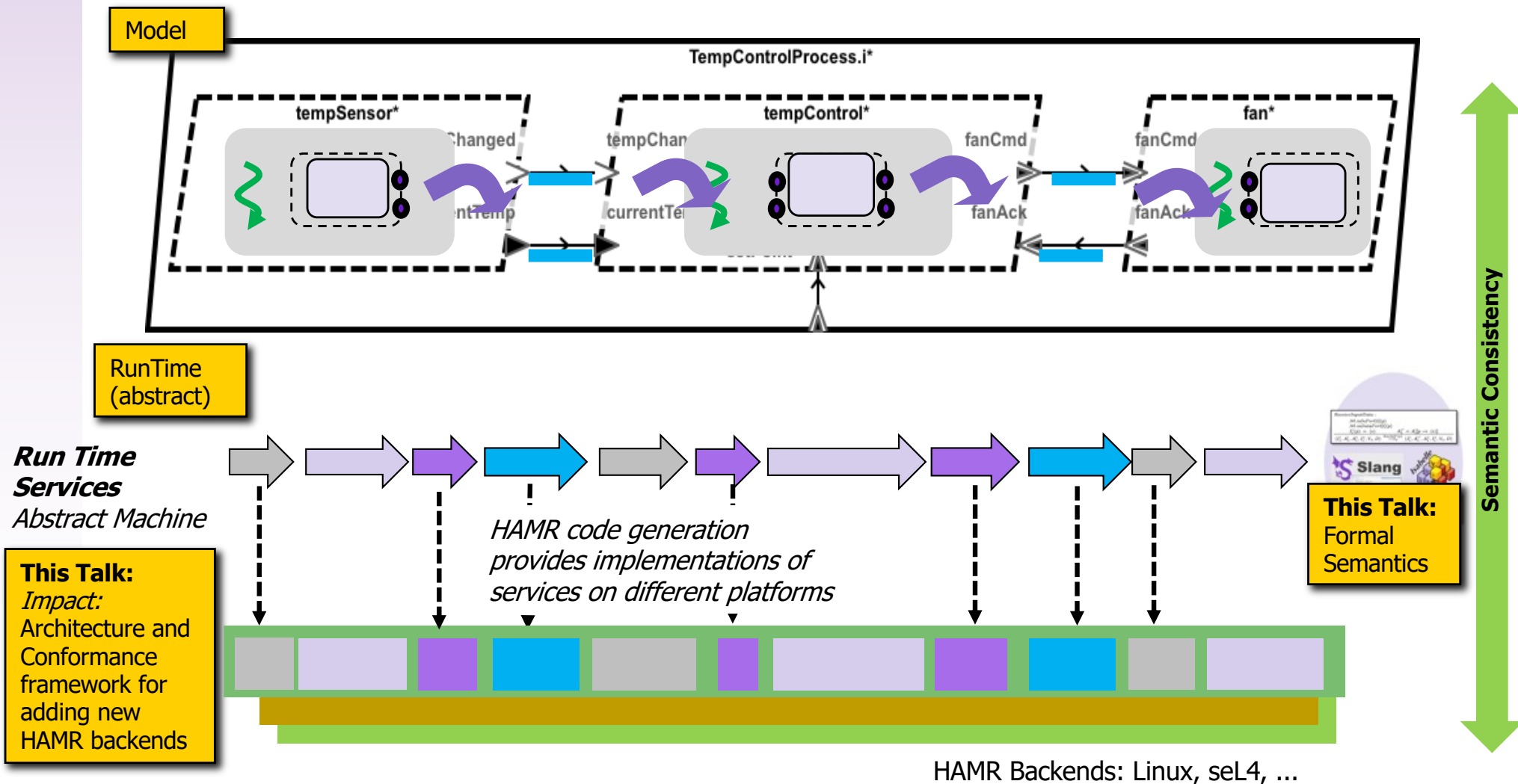
```
  -- List of ports whose input is frozen
```

```
end Receive_Input;
```

High-level API that **omits** specifics of what aspects of thread/system state are updated and the semantics of the updates...

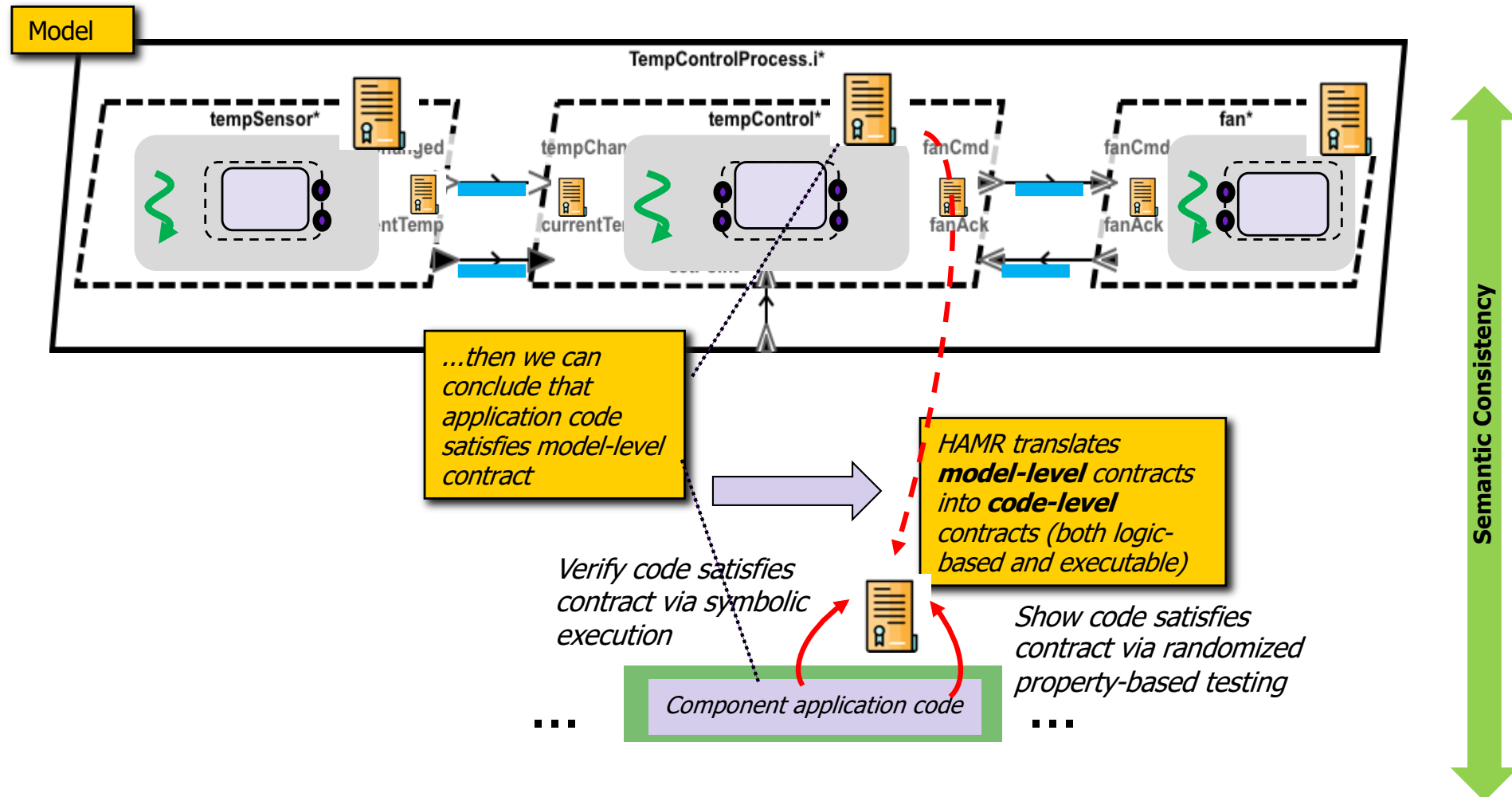
Formalizing AADL Run-Time Services

The formal semantics is a **specification for implementing AADL run-time on different platforms**

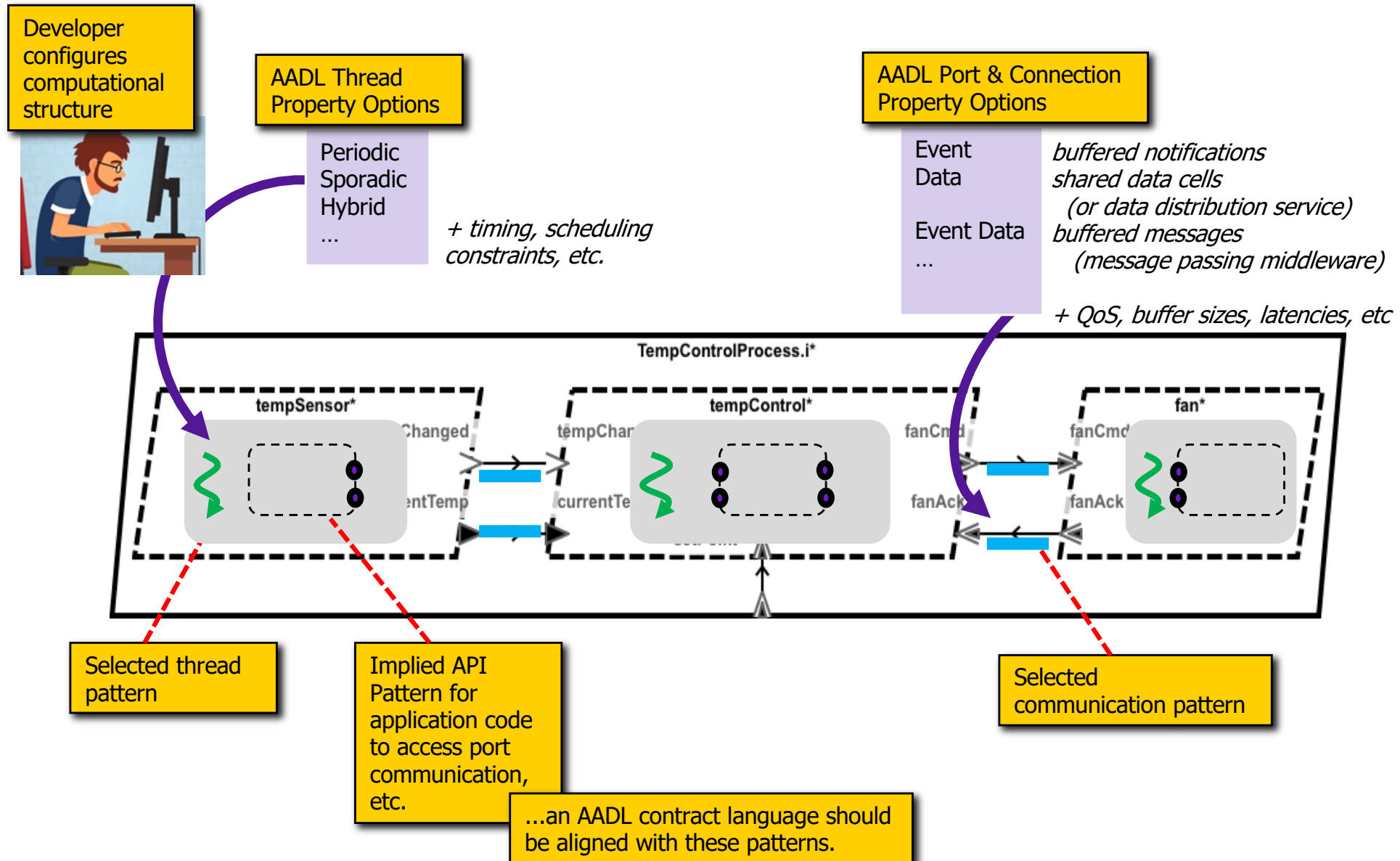


Formalizing AADL Run-Time Services

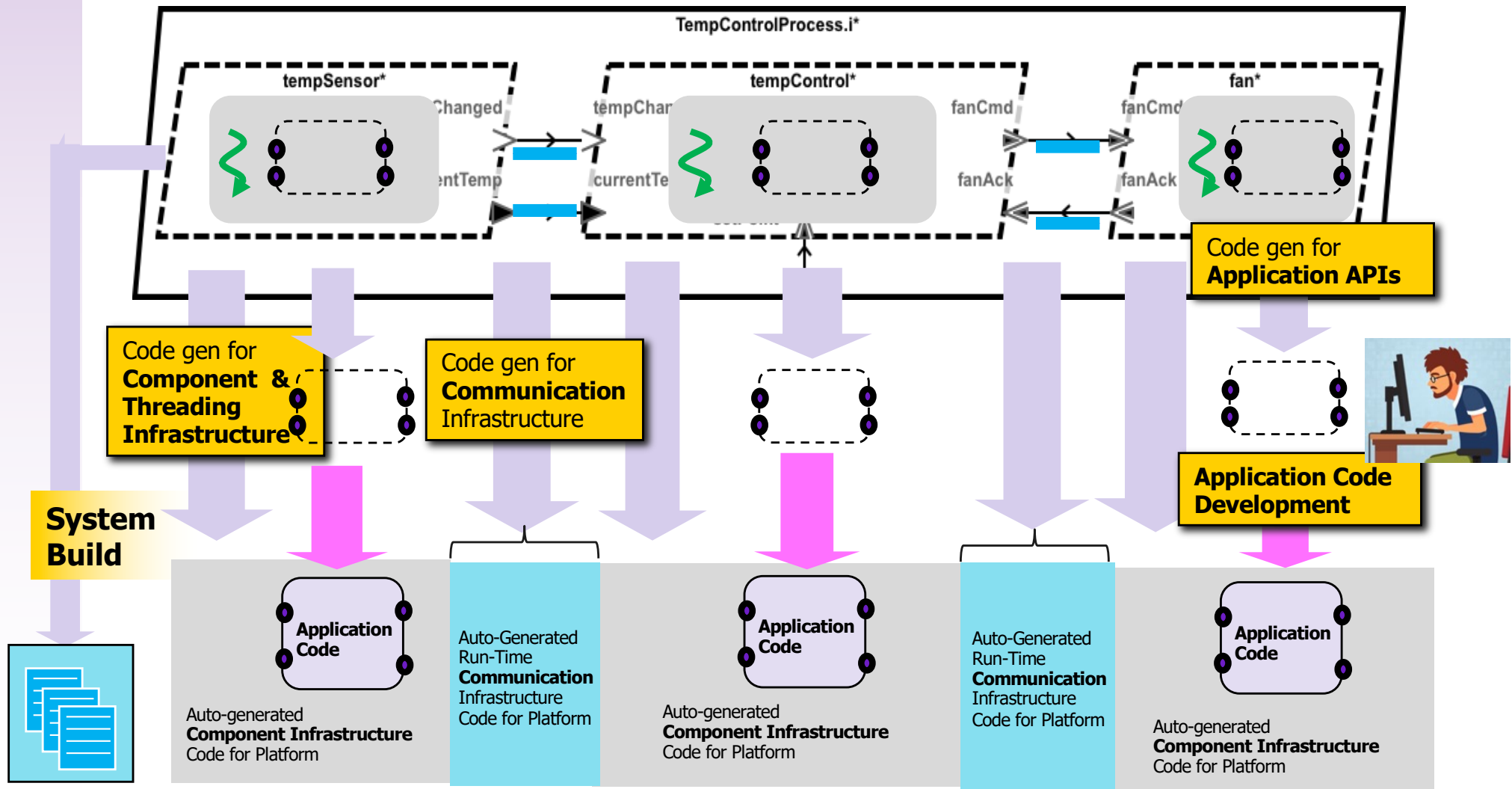
The formal semantics guides the design of an **integrated model and code-level contract framework** that supports both verification and property-based testing



AADL Modeling Concepts

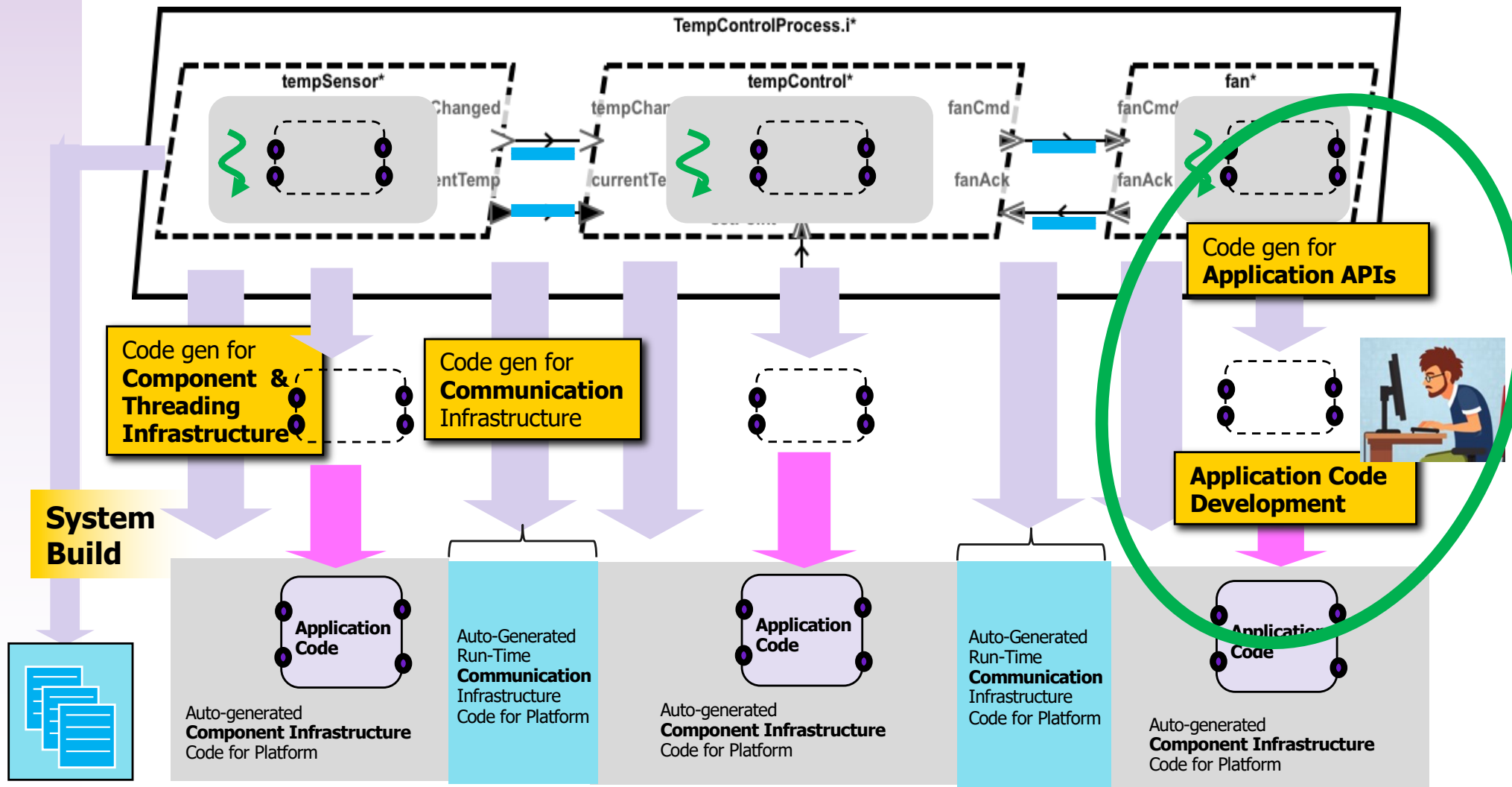


HAMR Code Generation



Platform configuration
information

HAMR Code Generation



Platform configuration
information

Component Application Code Interfaces Generated from AADL Model

Periodic Thread w/ data ports

AADL Model Implied Semantics

auto-generated

Application Code Skeleton in Slang

...Interfaces/APIs/Skeletons for application code are auto-generated from AADL model

Skeleton for application code entry point

```
source_impl_thermostat_regulate_temperature_manage_heat_source.scala [isolette_single_sensor_Instance]
regulate_temperature_manage_heat_s
Slang Script Runner
Git:
Manage_Heat_Source_impl_thermostat_regulate_temperature_manage_heat_source.scala
//=====
// Compute Entry Point
//=====
def timeTriggered(api: Manage_Heat_Source_impl_Operational_Api): Unit = {
  // add application code here
}

def activate(api: Manage_Heat_Source_impl_Operational_Api): Unit = { }

def deactivate(api: Manage_Heat_Source_impl_Operational_Api): Unit = { }

def finalise(api: Manage_Heat_Source_impl_Operational_Api): Unit = { }

def recover(api: Manage_Heat_Source_impl_Operational_Api): Unit = { }
```

Component Application Code Interfaces Generated from AADL Model

Periodic Thread w/ data ports

AADL Model Implied Semantics

auto-generated

Application Code Skeleton in Slang

...Interfaces/APIs/Skeletons for application code are auto-generated from AADL model

Adding application code to skeleton

```
def timeTriggered(api: Manage_Heat_Source_impl_Operational_Api): Unit = {  
  // ----- Get values of input ports -----  
  val lower: Isolette_Data_Model.Temp_impl = api.get_lower_desired_temp().get  
  val upper: Isolette_Data_Model.Temp_impl = api.get_upper_desired_temp().get  
  val regulator_mode: Isolette_Data_Model.Regulator_Mode.Type = api.get_regulator_mode().get  
  val currentTemp: Isolette_Data_Model.TempWstatus_impl = api.get_current_tempWstatus().get  
  
  //===== compute / control logic =====  
  var currentCmd: Isolette_Data_Model.On_Off.Type = lastCmd  
  regulator_mode match {...}  
  
  // ----- Set value of output port -----  
  api.put_heat_control(currentCmd)  
  lastCmd = currentCmd  
}  
  
def activate(api: Manage_Heat_Source_impl_Operational_Api): Unit = { }  
  
def deactivate(api: Manage_Heat_Source_impl_Operational_Api): Unit = { }  
  
def finalise(api: Manage_Heat_Source_impl_Operational_Api): Unit = { }
```

Component Application Code Interfaces Generated from AADL Model

Periodic Thread w/ data ports

heat_control Put

regulator_mode Get

AADL Model Implied Semantics

auto-generated

Application Code Skeleton in Slang

...Interfaces/APIs/Skeletons for application code are auto-generated from AADL model

Reading a value from the **regulator_mode** input data port using auto-generated API

```
def timeTriggered(api: Manage_Heat_Source_impl_Operational_Api): Unit = {  
  // ----- Get values of input ports -----  
  val lower: Isolette_Data_Model.Temp_impl = api.get_lower_desired_temp().get  
  val upper: Isolette_Data_Model.Temp_impl = api.get_upper_desired_temp().get  
  val regulator_mode: Isolette_Data_Model.Regulator_Mode.Type = api.get_regulator_mode().get  
  val currentTemp: Isolette_Data_Model.TempWstatus_impl = api.get_current_tempwstatus().get  
  
  //===== compute / control logic =====  
  var currentCmd: Isolette_Data_Model.On_Off.Type = lastCmd  
  regulator_mode match {...}  
  
  // ----- Set value of output port -----  
  api.put_heat_control(currentCmd)  
  lastCmd = currentCmd  
}
```

Putting a value from the **heat_control** output data port using auto-generated API

AADL Port and Thread Execution Semantics



"Analyzable Real-Time Systems"
Burns & Wellings

On each dispatch, AADL threads follow a well-known **input-compute-output** pattern for real-time tasks that aid analysis and verification...

(1) Receive inputs

Communication Infrastructure - **Inputs**

AADL Component Application Memory Boundary

Input Application Port State

Application Code

Output Application Port State

gets

(2) Compute, Run to Completion

puts

Communication Infrastructure - **Outputs**

(3) Send outputs

From AADL standard...

(2) AADL supports an input-compute-output model of communication and execution for threads and port-based communication. The inputs received from other components are frozen at a specified point, by default the dispatch of a thread. As a result the

AADL Port and Thread Execution Semantics



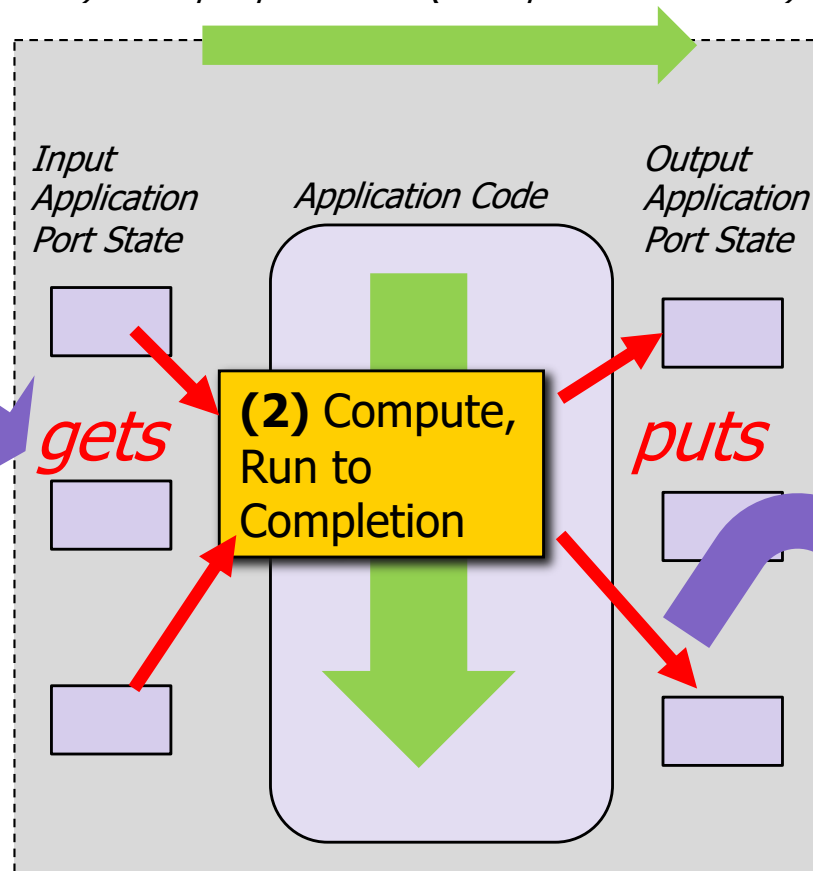
"Analyzable Real-Time Systems"
Burns & Wellings

On each dispatch, AADL threads follow a well-known **input-compute-output** pattern for real-time tasks that aid analysis and verification...

(1) Receive inputs

Communication Infrastructure - **Inputs**

Abstractly, a function from input ports states (and local data) to output port states (and updated local data)



Communication Infrastructure - **Outputs**

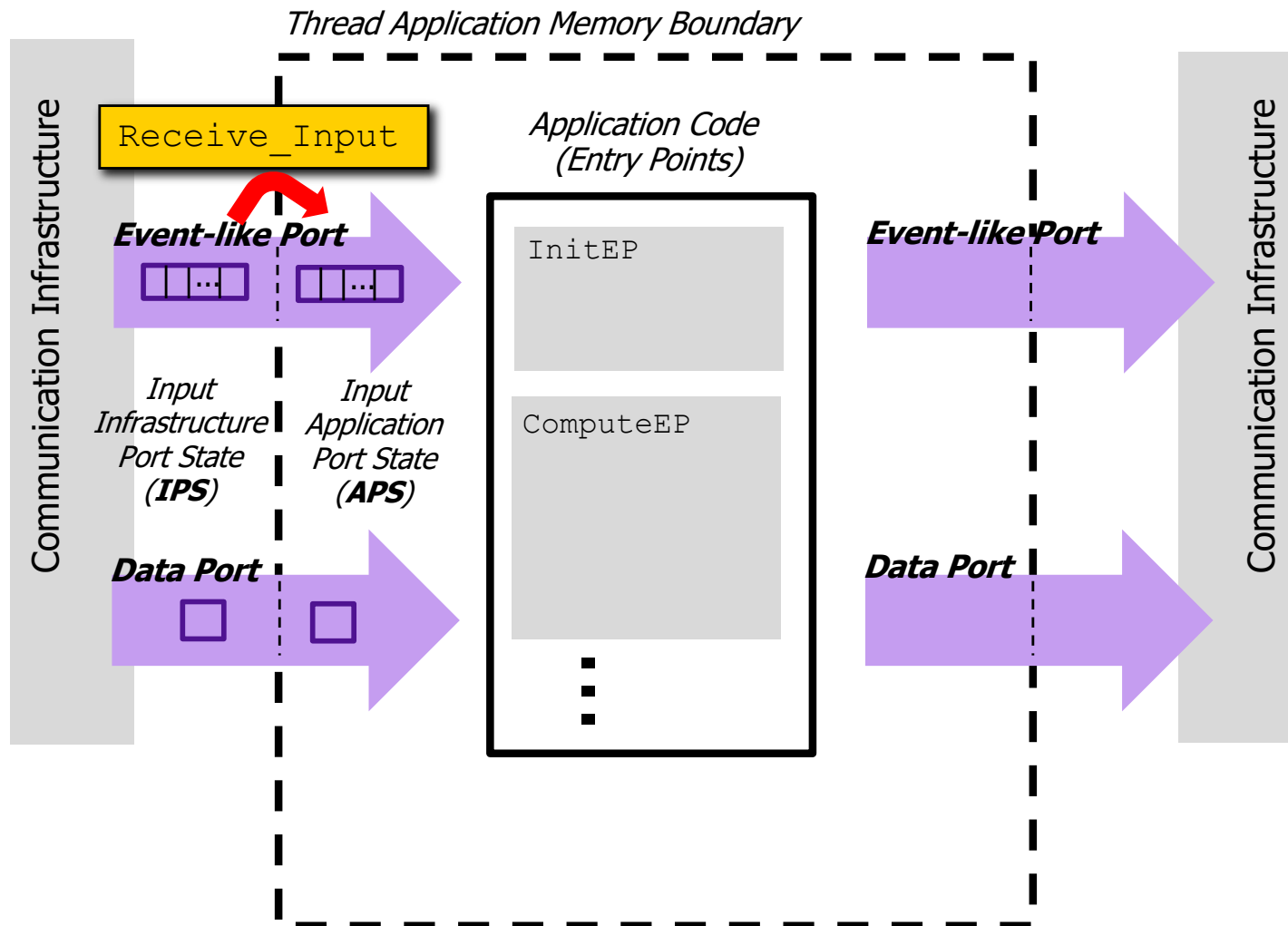
(3) Send outputs

From AADL standard...

(2) AADL supports an input-compute-output model of communication and execution for threads and port-based communication. The inputs received from other components are frozen at a specified point, by default the dispatch of a thread. As a result the

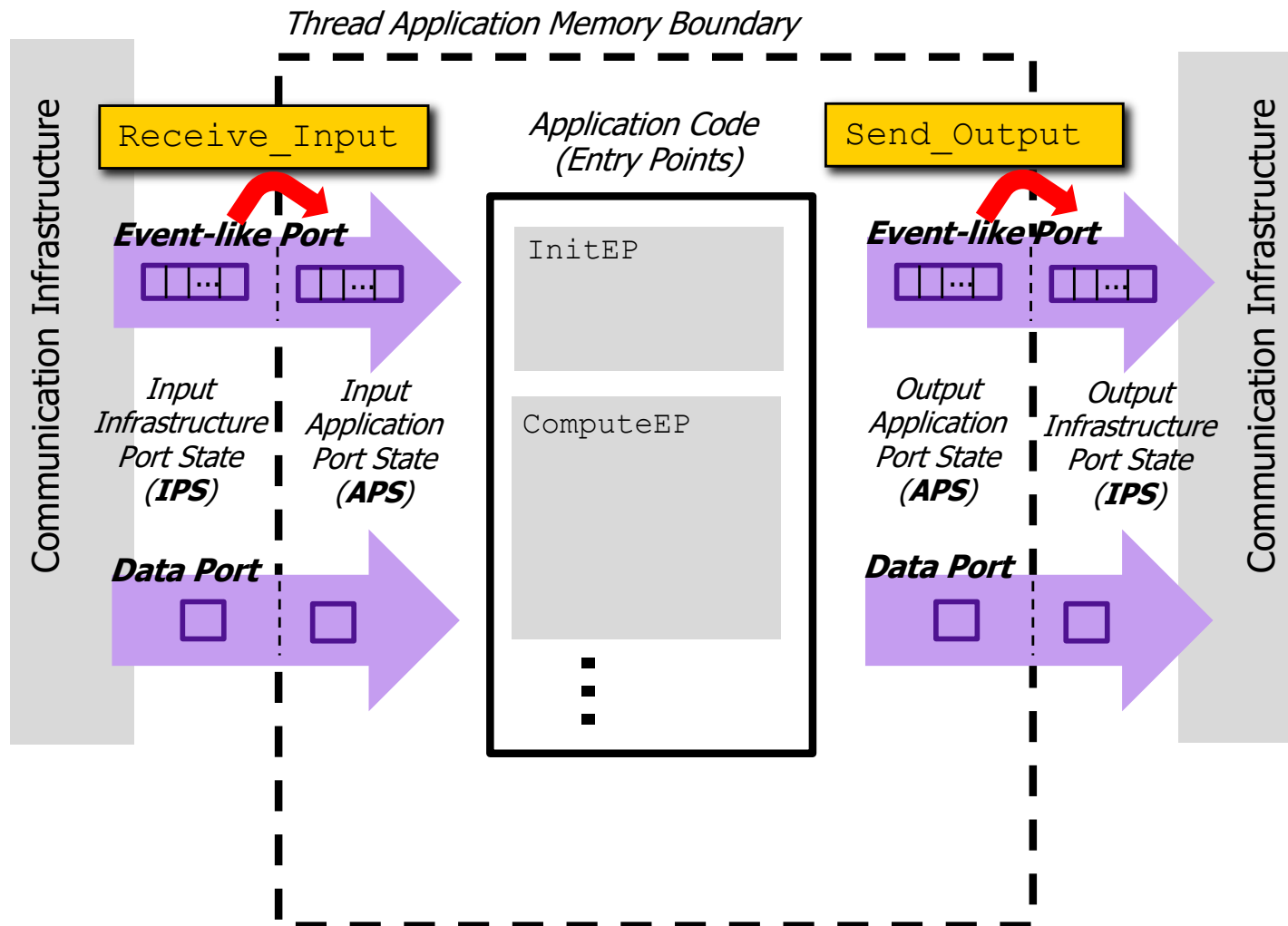
Key Concepts

Towards Formalism: Clarify key elements of the thread state and the run-time service operations on elements



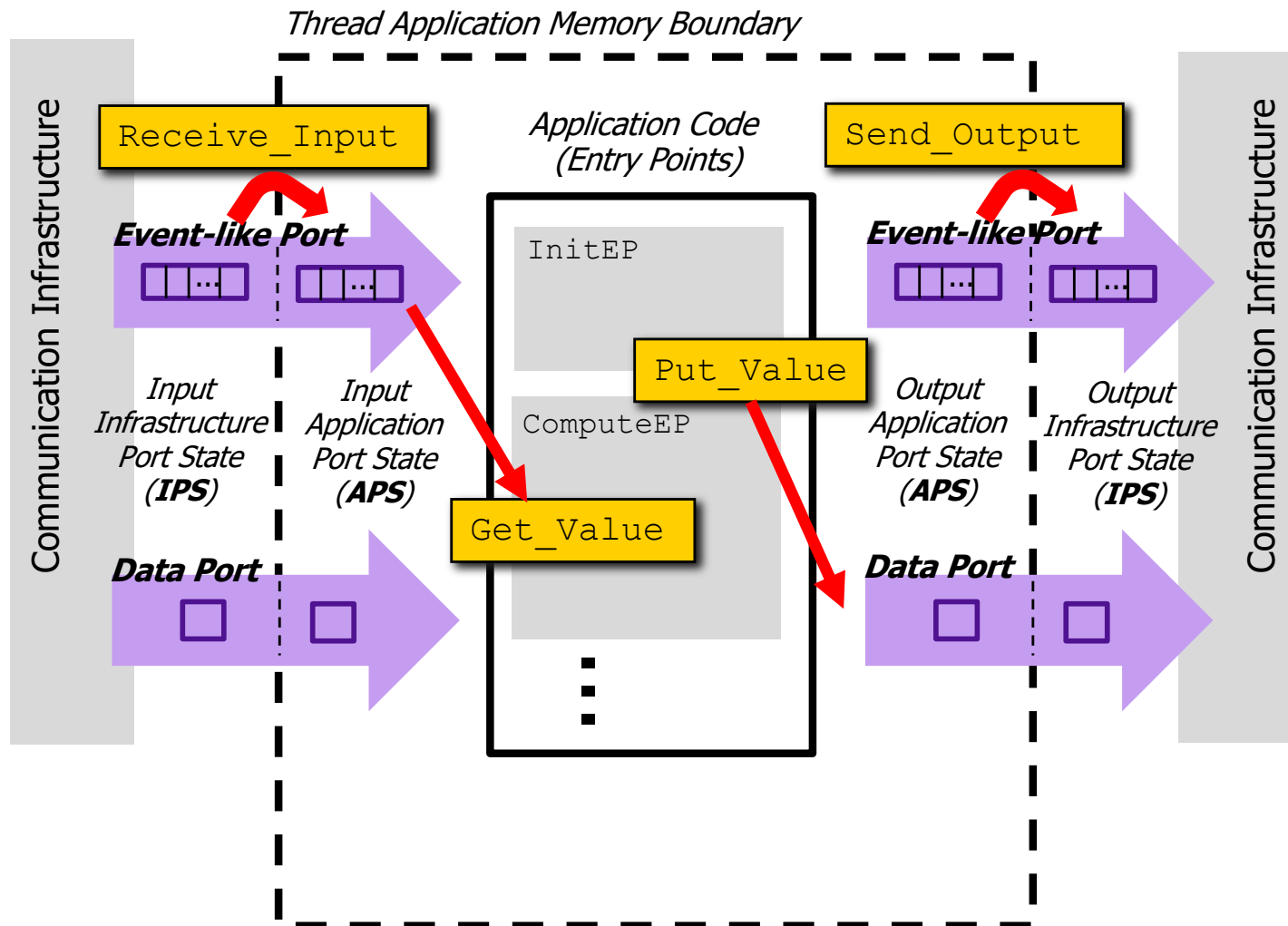
Key Concepts

Towards Formalism: Clarify key elements of the thread state and the run-time service operations on elements



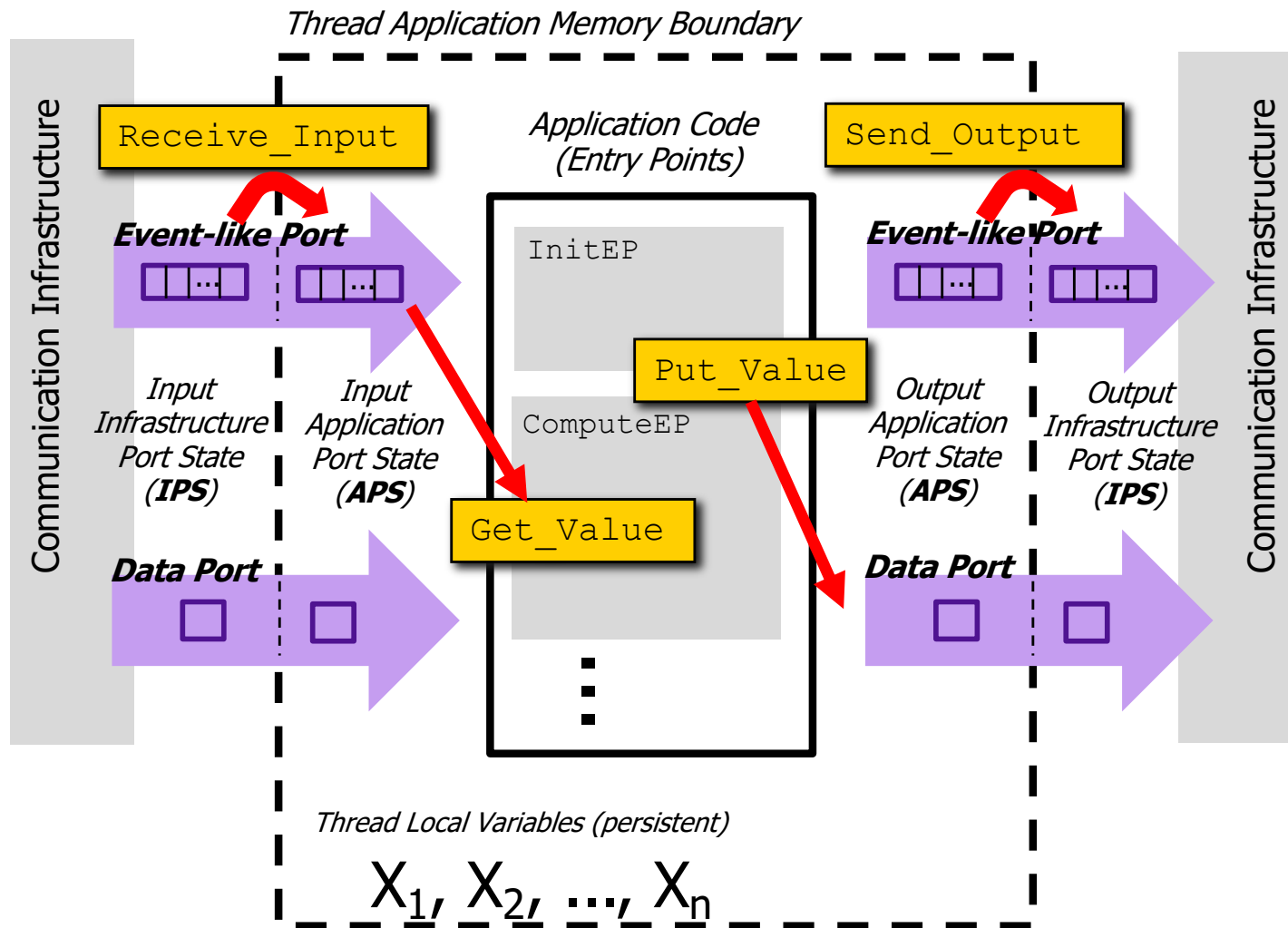
Key Concepts

Towards Formalism: Clarify key elements of the thread state and the run-time service operations on elements



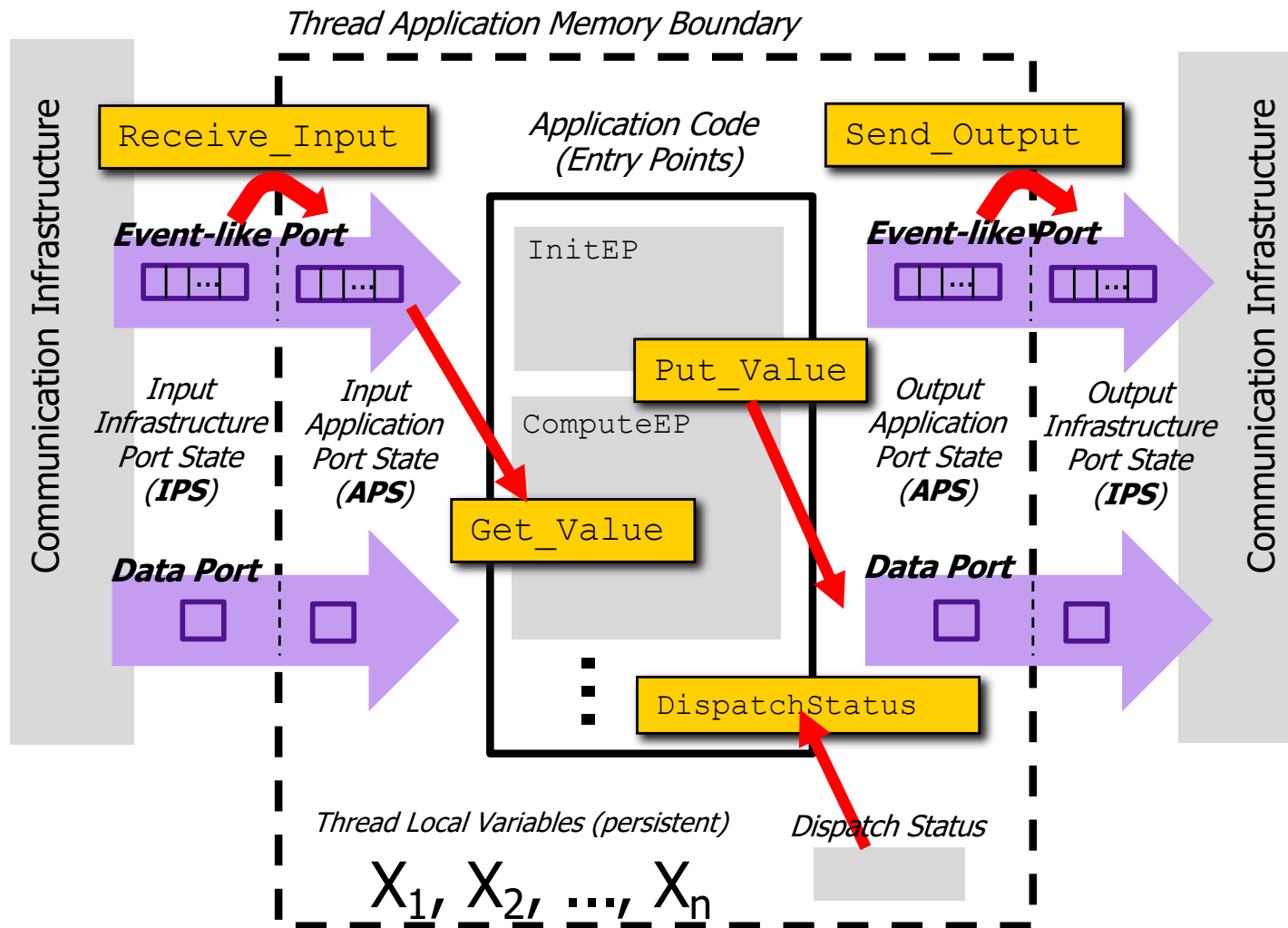
Key Concepts

Towards Formalism: Clarify key elements of the thread state and the run-time service operations on elements



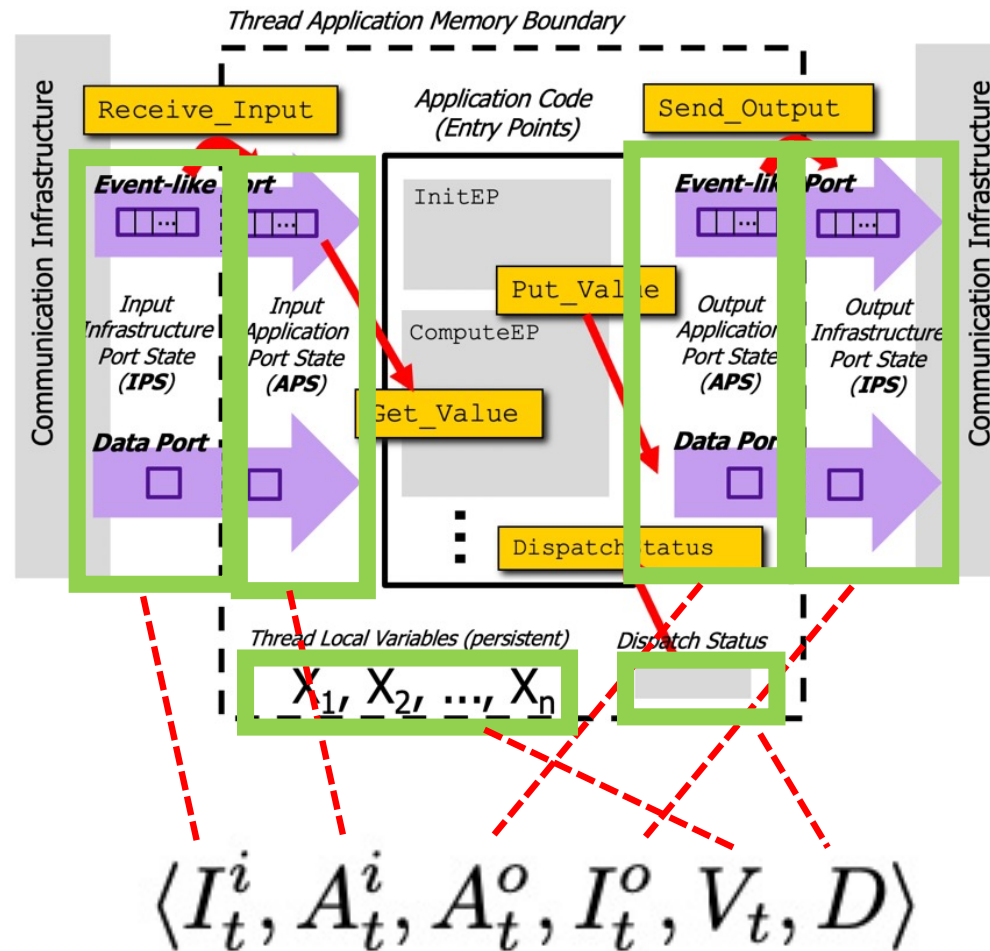
Key Concepts

Towards Formalism: Clarify key elements of the thread state and the run-time service operations on elements



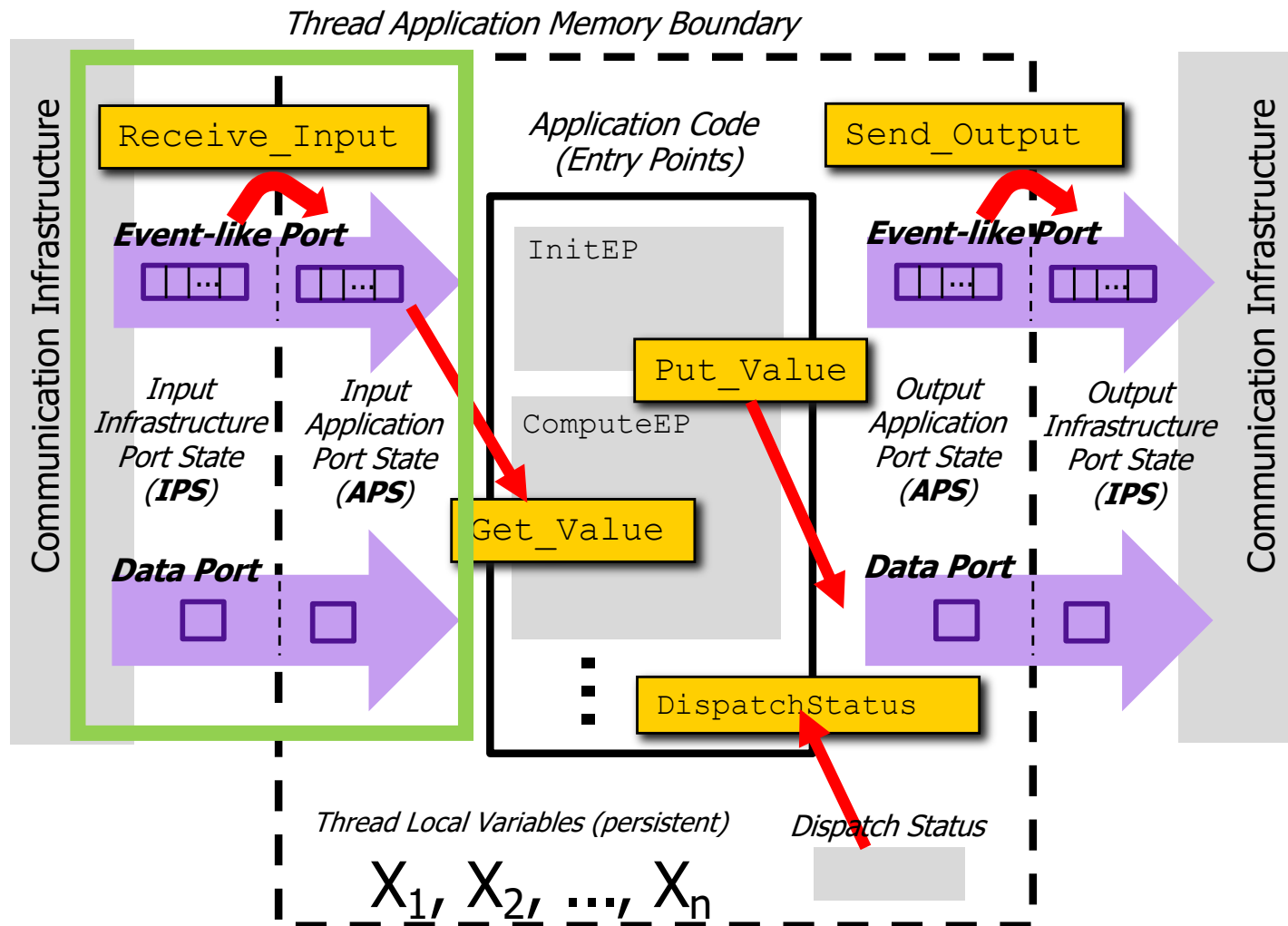
Formalization of Thread State

The concepts of state in this diagram now become part of the formalization of a thread state...



Receive Input Runtime Service

Let's consider the formalization for Receive Input run-time service...



Receive Input Runtime Service

Rules formalizing the behavior of **Receive Input** runtime service

Formalization clarifies that on these portions of the thread state are modified by the service.

ReceiveInputData :

$$\frac{\begin{array}{l} \mathcal{M}.isInPort[t](p) \\ \mathcal{M}.isDataPort[t](p) \\ I_t^i(p) = \langle v \rangle \end{array} \quad A_t^{i'} = A_t^i[p \mapsto \langle v \rangle]}{\langle I_t^i, A_t^i, A_t^o, I_t^o, V_t, D \rangle \xrightarrow{t, \text{RecInP}(p)} \langle I_t^i, A_t^{i'}, A_t^o, I_t^o, V_t, D \rangle}$$

ReceiveInputEventLikeOneItem :

$$\frac{\begin{array}{l} \mathcal{M}.isInPort[t](p) \\ \mathcal{M}.isEventLikePort[t](p) \\ \mathcal{M}.DequeuePolicy[t](p) = \text{OneItem} \\ I_t^i(p) = \langle q \triangleright v \rangle \end{array} \quad I_t^{i'} = I_t^i[p \mapsto q] \quad A_t^{i'} = A_t^i[p \mapsto \langle v \rangle]}{\langle I_t^i, A_t^i, A_t^o, I_t^o, V_t, D \rangle \xrightarrow{t, \text{RecInP}(p)} \langle I_t^{i'}, A_t^{i'}, A_t^o, I_t^o, V_t, D \rangle}$$

ReceiveInputEventLikeAllItems :

$$\frac{\begin{array}{l} \mathcal{M}.isInPort[t](p) \\ \mathcal{M}.isEventLikePort[t](p) \\ \mathcal{M}.DequeuePolicy[t](p) = \text{AllItems} \\ \neg isEmpty(I_t^i(p)) \end{array} \quad I_t^{i'} = I_t^i[p \mapsto \langle \rangle] \quad A_t^{i'} = A_t^i[p \mapsto I_t^i(p)]}{\langle I_t^i, A_t^i, A_t^o, I_t^o, V_t, D \rangle \xrightarrow{t, \text{RecInP}(p)} \langle I_t^{i'}, A_t^{i'}, A_t^o, I_t^o, V_t, D \rangle}$$

ReceiveInputEventLikeEmpty :

$$\frac{\begin{array}{l} \mathcal{M}.isInPort[t](p) \\ \mathcal{M}.isEventLikePort[t](p) \\ isEmpty(I_t^i(p)) \end{array} \quad A_t^{i'} = A_t^i[p \mapsto \langle \rangle]}{\langle I_t^i, A_t^i, A_t^o, I_t^o, V_t, D \rangle \xrightarrow{t, \text{RecInP}(p)} \langle I_t^i, A_t^{i'}, A_t^o, I_t^o, V_t, D \rangle}$$

Different behaviors for *data* ports vs *event data* ports

Rules take into account different dequeuing policies configured by AADL properties

Artifact Correspondence/Traceability

Slang Executable Semantics: There is a 1-to-1 correspondence between the mathematical definition of the state and the representation in the executable specification...

$$\langle I_t^i, A_t^i, A_t^o, I_t^o, V_t, D \rangle$$

```
@record class ThreadState(  
    IIn: PortStates,  
    AIn: PortStates,  
    AOut: PortStates,  
    IOut: PortStates,  
    V: VarStates,  
    dispatchStatus: Option[DispatchLogic.DispatchStatusInfo]  
)
```

...from Executable Functional Model in *Slang*

...in the executable model, simple logging gives a nice way to see the state transitions of the semantics

Receive Input Runtime Service

Executable functional specification

```
def receive_input_onePort(portId: Model.PortId, iin: PortStates, ain: PortStates) : (PortStates, PortStates) = {
  Model.getPortKind(portId) match {
    // ----- Data Ports -----
    case Model.PortProperties.Kind.Data =>
      // == Latex Rule: ReceiveInputData
      // infrastructure ports (i in)
      // -- leave values unchanged - next dispatch can read old value
      val iin_new : PortStates = iin
      // application port (a in)
      // - "freeze value" of port by copying value from infrastructure port to port variable
      val queueValue: PortQueue.Type = iin.get(portId).get // get current infrastructure port value
      assert(queueValue.size == 1) // the size of a data port queue should always be 1
      val ain_new : PortStates = ain + portId ~> queueValue // copy current value to port variable
      return (iin_new, ain_new)
    // ----- Event Ports -----
    case Model.PortProperties.Kind.Event =>
      val iin_portId_queue: PortQueue.Type = iin.get(portId).get
      // val ain_port: PortQueue.Type = ain.get(portId).get // consider seeding the helper below with empty queue
      val (iin_portId_queue_new, ain_portId_queue_new) = receive_input_event_port_dequeue_policy(iin_portId_queue)
      /* iin_new queue is emptied for this port */
      val iin_new = iin + portId ~> iin_portId_queue_new
      val ain_new = ain + portId ~> ain_portId_queue_new
      return (iin_new, ain_new)
  }
}
```

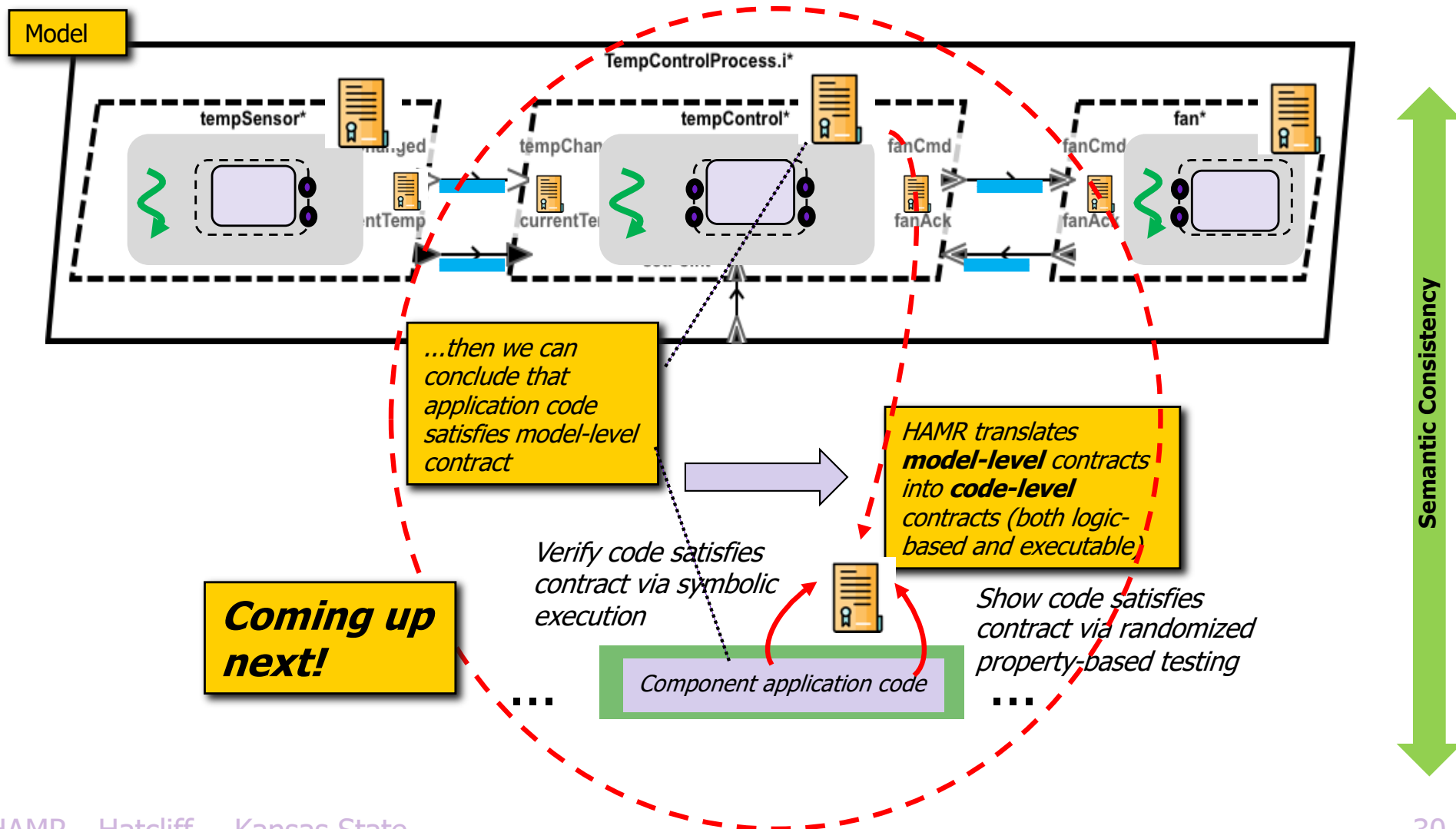
Code clarifies that only these parts of the thread state are modified by the service.

Different behaviors for *data* ports vs *event* ports

Following the rules, the code takes into account different dequeuing policies configured by AADL properties

Application: Contracts

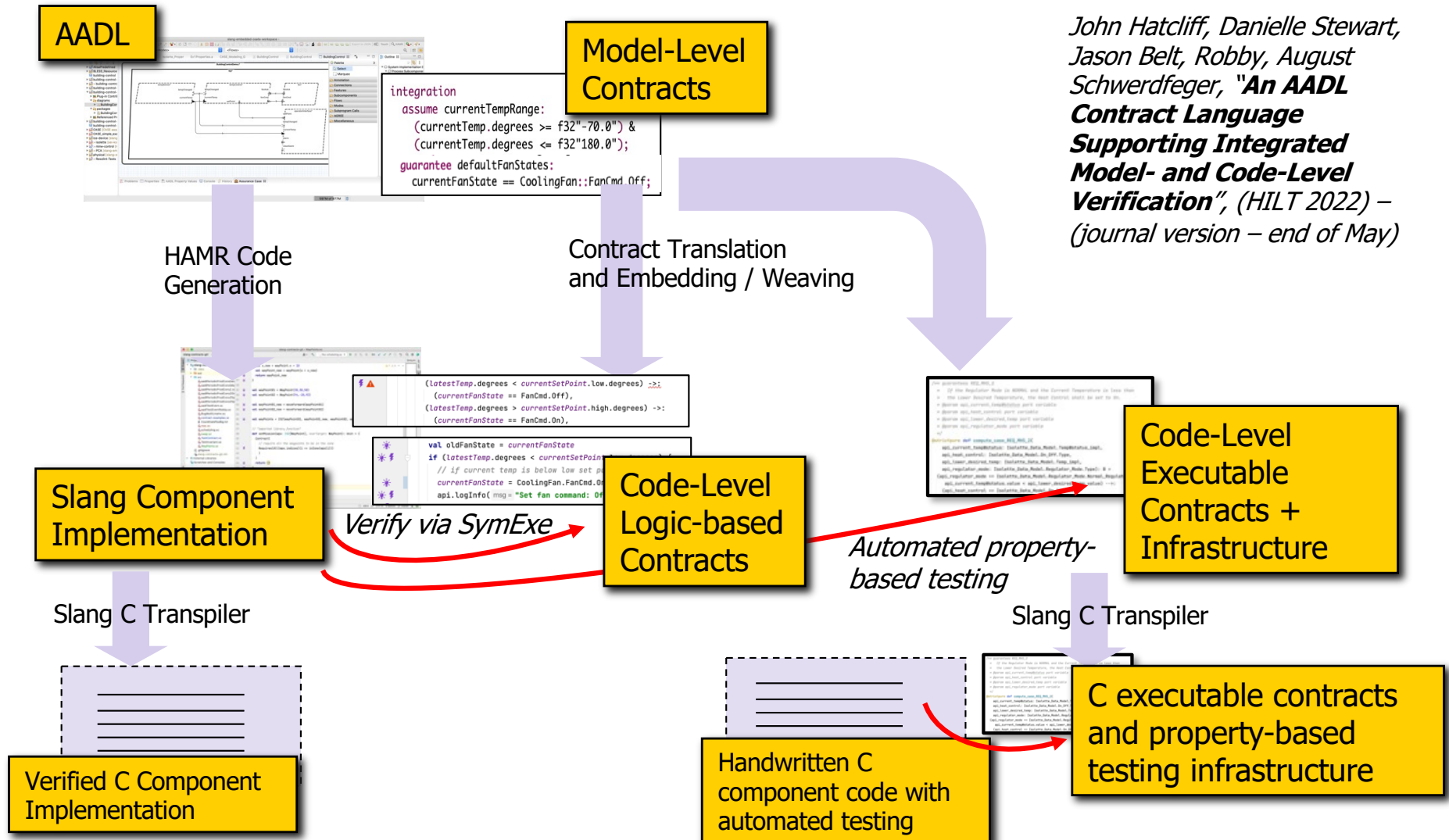
The formal semantics guides the design of an **integrated model and code-level contract framework** that supports both verification and property-based testing



Application: *Integrated Model/Code Contract Language*

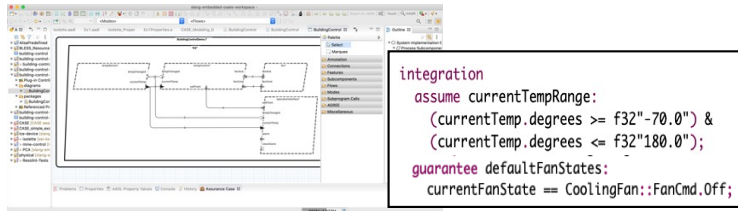
KSU / Adventium Labs (now Galois) ARMY SBIR Phase II...

John Hatcliff, Danielle Stewart,
Jason Belt, Robby, August
Schwerdfeger, **"An AADL
Contract Language
Supporting Integrated
Model- and Code-Level
Verification"**, (HILT 2022) –
(journal version – end of May)



GUMBO - AADL Contract Language

GUMBO Contract Language Features



*Inspired by previous work
on AGREE and BLESS*

*Example in
this talk*

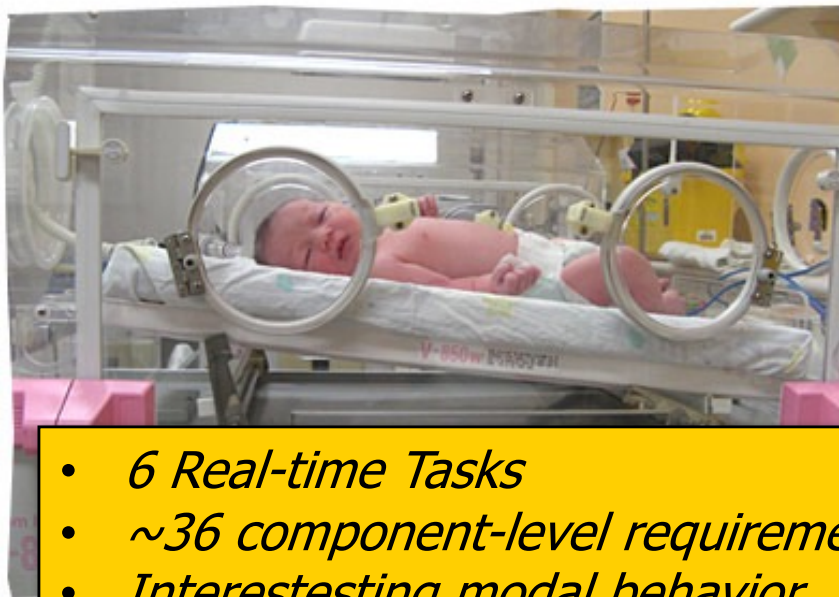
- Data type invariants
- Port invariants (integration constraints)
- Event-based / Shared-data based inter-thread communication
- Local state declarations with invariants
- Pre/Post conditions for AADL thread code entrypoints
 - Initialize Entry Point
 - Compute Entry Point
 - Periodic
 - Sporadic (collection of event handlers)
- Support for fixed width scalars (e.g., Float32)
- Support for almost all of the AADL Data Modeling Annex

FAA Requirements Engineering Management Handbook (REMH)

REMH

Illustrate with the Isolette example from FAA REMH

- Written for the FAA by engineers at Rockwell Collins (David L. Lempia, Steven P. Miller)
- Includes example of an “Isolette” (infant incubator)



- *6 Real-time Tasks*
- *~36 component-level requirements*
- *Interestesting modal behavior*

DOT/FAA/AR-08/32

Air Traffic Organization
NextGen & Operations Planning
Office of Research and
Technology Development
Washington, DC 20591

Requirements Engineering Management Handbook

June 2009

Final Report

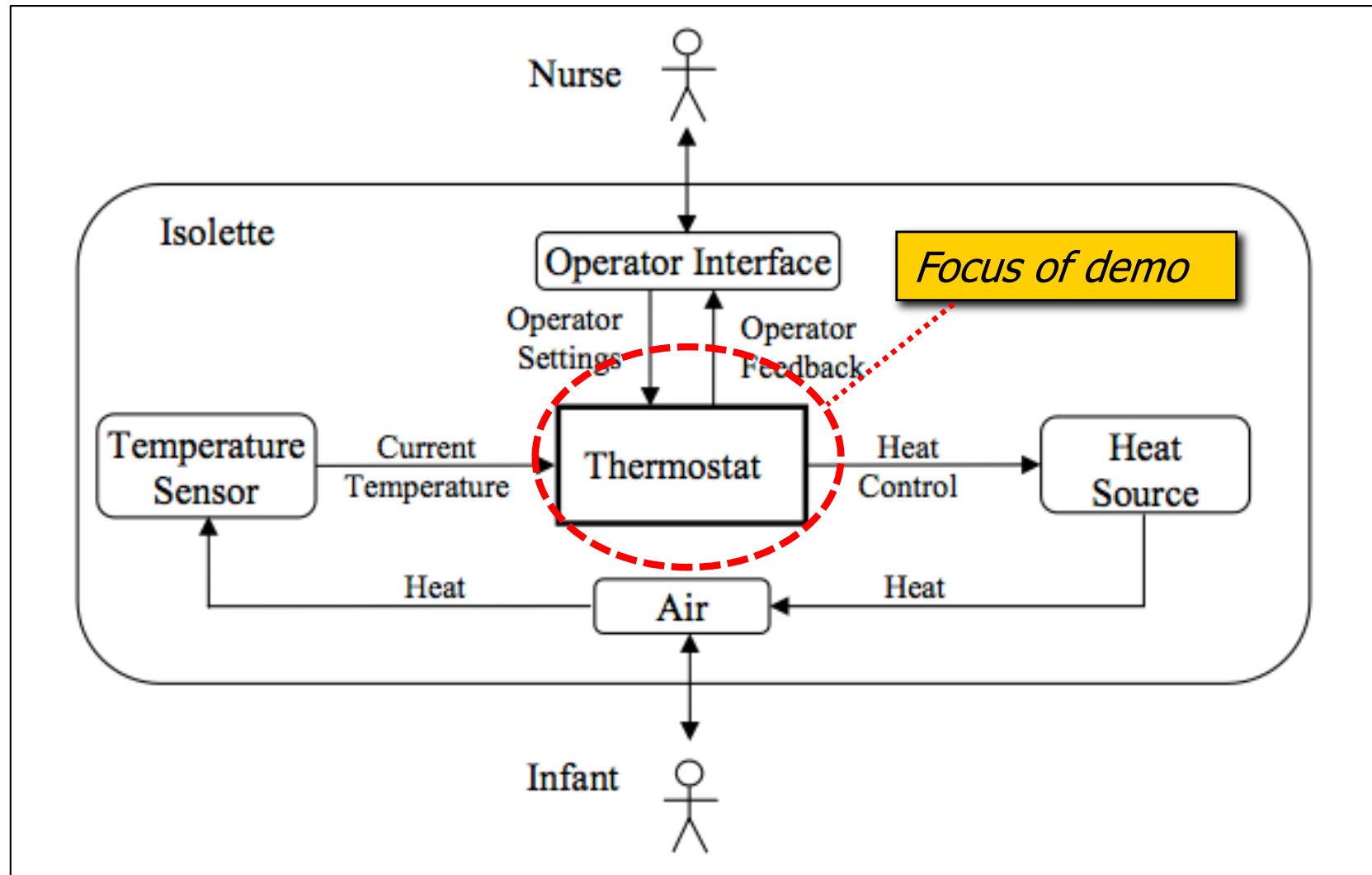
This document is available to the U.S. public through
the National Technical Information Service (NTIS),
Springfield, Virginia 22161.



U.S. Department of Transportation
Federal Aviation Administration

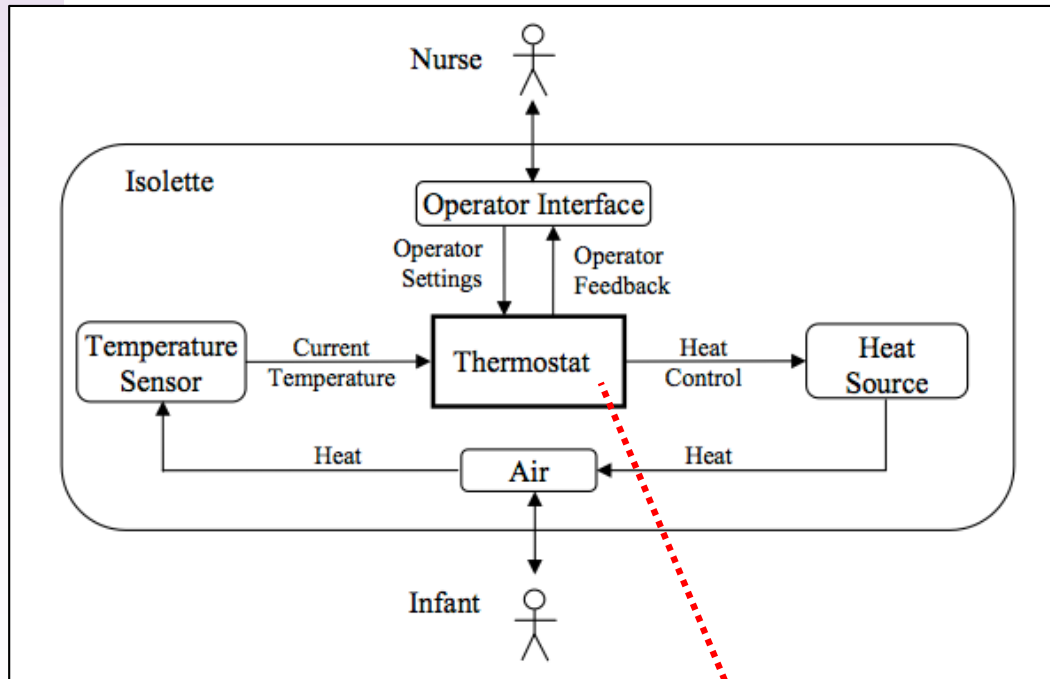
Focus of Example

Isolette Thermostat – heat controller for incubator

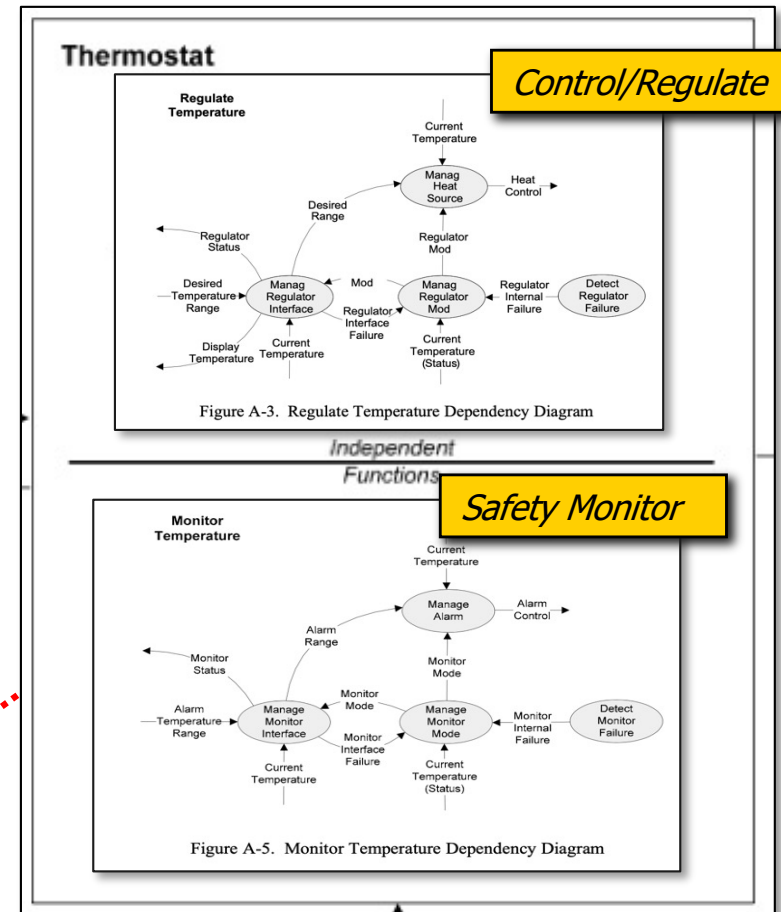


Decomposing Thermostat

The FAA REMH decomposes the Isolette into a control system and safety monitor with three tasks each



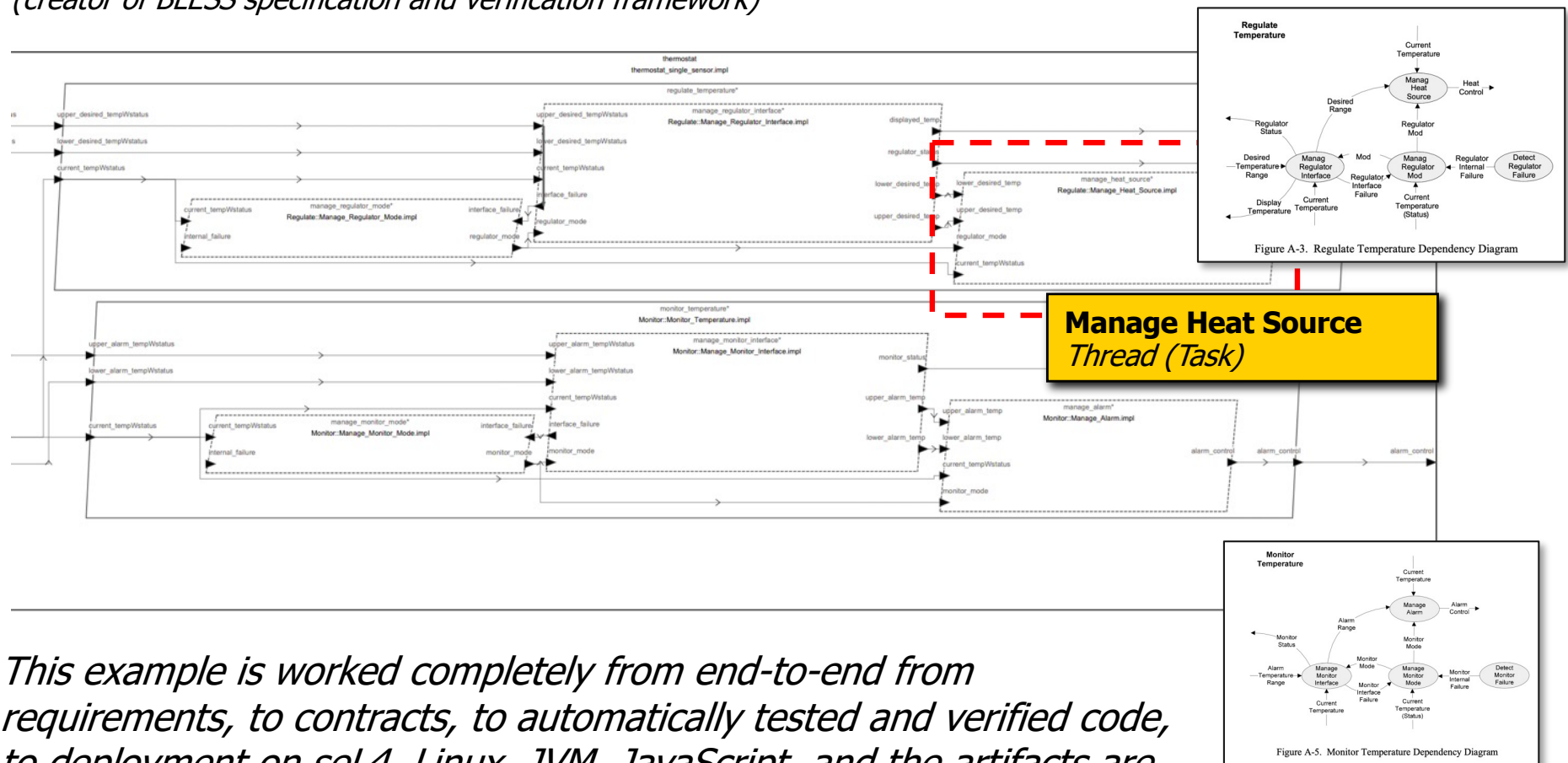
Decomposing the Thermostat into Regulate Temperature and Monitor Temperature functions.



Using AADL to Represent Design

AADL Model

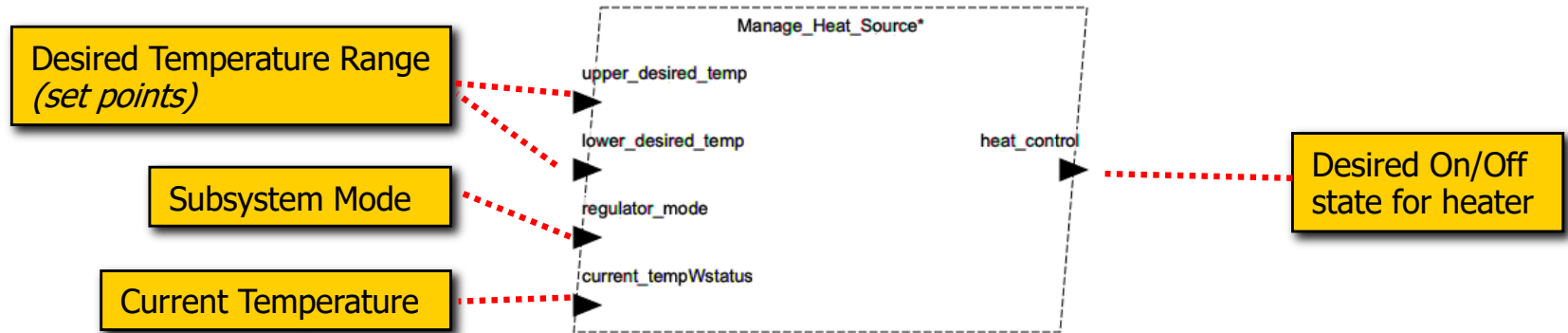
*AADL model originally developed by Brian Larson
(creator of BLESS specification and verification framework)*



This example is worked completely from end-to-end from requirements, to contracts, to automatically tested and verified code, to deployment on seL4, Linux, JVM, JavaScript, and the artifacts are publicly available.

Manage Heat Source Thread

AADL Interface for **Manage Heat Source** Thread



thread Manage_Heat_Source
features

```
-- ===== INPUTS =====
-- ("Current Temperature") - current temperature (from temp sensor)
current_tempWstatus: in data port Isolette_Data_Model::TempWstatus.impl;
-- ("Desired Range") - lowest and upper bound of desired temperature range
lower_desired_temp: in data port Isolette_Data_Model::Temp.impl;
upper_desired_temp: in data port Isolette_Data_Model::Temp.impl;
-- ("Regulator Mode") - subsystem mode
regulator_mode: in data port Isolette_Data_Model::Regulator_Mode;

-- ===== OUTPUTS =====
-- ("Heat Control") - command to turn heater on/off (actuation command)
heat_control: out data port Isolette_Data_Model::On_Off;
```

Requirements to Contracts

FAA REMH requirements for **Manage Heat Source** task

DOT/FAA/AR-08/32
Air Traffic Organization
NextGen & Operations Planning
Office of Research and
Technology Development
Washington, DC 20591

Requirements Engineering
Management Handbook

Requirements for control laws of this task...

REQ-MHS-1: If the Regulator Mode is INIT, the Heat Control shall be set to Off.

Rationale: A regulator that is initializing cannot regulate the Current Temperature of the Isolette and the Heat Control should be turned off.

REQ-MHS-2: If the Regulator Mode is NORMAL and the Current Temperature is less than the Lower Desired Temperature, the Heat Control shall be set to On.

REQ-MHS-3: If the Regulator Mode is NORMAL and the Current Temperature is greater than the Upper Desired Temperature, the Heat Control shall be set to Off.

REQ-MHS-4: If the Regulator Mode is NORMAL and the Current Temperature is greater than or equal to the Lower Desired Temperature and less than or equal to the Upper Desired Temperature, the value of the Heat Control shall not be changed.

REQ-MHS-5: If the Regulator Mode is FAILED, the Heat Control shall be set to Off.

Available to the U.S. public through
Information Service (NTIS),
22161.

Transportation
Administration

Requirements to Contracts

GUMBO contracts are written together with the thread interface in the AADL OSATE IDE (using AADL Annex clause)

```
400
401 thread Manage_Heat_Source
402 features
403   -- ===== INPUTS =====
404   -- ("Current Temperature") - current temperature (from temp sensor)
405   current_tempWstatus: in data port Isolette_Data_Model::TempWstatus.impl;
406   -- ("Desired Range") - lowest and upper bound of desired temperature range
407   lower_desired_temp: in data port Isolette_Data_Model::Temp.impl;
408   upper_desired_temp: in data port Isolette_Data_Model::Temp.impl;
409   -- ("Regulator Mode") - subsystem mode
410   regulator_mode: in data port Isolette_Data_Model::Regulator_Mode;
411
412   -- ===== OUTPUTS =====
413   -- ("Heat Control") - command to turn heater on/off (actuation command)
414   heat_control: out data port Isolette_Data_Model::On_Off;
415
416 properties
417   Dispatch_Protocol => Periodic;
418   Period => Isolette_Properties::ThreadPeriod;
419
420   Stack_Size => Isolette_Properties::StackSize;
421
422 annex GUMBO {
423   -- indicate that the component maintains an internal state (variables) that influence its behavior
424   state
425     lastCmd: Isolette_Data_Model::On_Off;
426
427   -- ===== Initialize Entry Point Behavior Constraints =====
428   initialize
429     guarantee
430       initlastCmd: lastCmd == Isolette_Data_Model::On_Off.Off;
431     REQ_MHS_1 "If the Regulator Mode is INIT, the Heat Control shall be
432       lset to Off";
433     heat_control == Isolette_Data_Model::On_Off.Off;
434
435   -- ===== Compute Entry Point Behavior Constraints =====
436   compute
437     -- assumption on set points enforced within the Operator Interface
438     assume lower_is_lower_temp: lower_desired_temp.value <= upper_desired_temp.value;
```

REQ-MHS-1: If the Regulator Mode is INIT, the Heat Control shall be set to Off.
Rationale: A regulator that is initializing cannot regulate the Current Temperature of the Isolette and the Heat Control should be turned off.

REQ-MHS-2: If the Regulator Mode is NORMAL and the Current Temperature is less than the Lower Desired Temperature, the Heat Control shall be set to On.

REQ-MHS-3: If the Regulator Mode is NORMAL and the Current Temperature is greater than the Upper Desired Temperature, the Heat Control shall be set to Off.

REQ-MHS-4: If the Regulator Mode is NORMAL and the Current Temperature is greater than or equal to the Lower Desired Temperature and less than or equal to the Upper Desired Temperature, the value of the Heat Control shall not be changed.

REQ-MHS-5: If the Regulator Mode is FAILED, the Heat Control shall be set to Off.



Developer
formalizes
requirements

Component
contract

Manage Heat Source Contracts

AADL GUMBO Contracts for **Manage Heat Source** Thread, with traceability to REMH requirements.

```
case REQ_MHS_1 "If the Regulator Mode is INIT, the Heat Control shall be
  lset to Off.":
  assume regulator_mode == Isolette_Data_Model::Regulator_Mode.Init_Regulator_Mode;
  guarantee heat_control == Isolette_Data_Model::On_Off.Off;

case REQ_MHS_2 "If the Regulator Mode is NORMAL and the Current Temperature is less than
  lthe Lower Desired Temperature, the Heat Control shall be set to On.":
  assume (regulator_mode == Isolette_Data_Model::Regulator_Mode.Normal_Regulator_Mode)
    & (current_tempWstatus.value < lower_desired_temp.value);
  guarantee heat_control == Isolette_Data_Model::On_Off.Onn;

case REQ_MHS_3 "If the Regulator Mode is NORMAL and the Current Temperature is greater than
  lthe Upper Desired Temperature, the Heat Control shall be set to Off.":
  assume (regulator_mode == Isolette_Data_Model::Regulator_Mode.Normal_Regulator_Mode)
    & (current_tempWstatus.value > upper_desired_temp.value);
  guarantee heat_control == Isolette_Data_Model::On_Off.Off;

case REQ_MHS_4 "If the Regulator Mode is NORMAL and the Current
  lTemperature is greater than or equal to the Lower Desired Temperature
  land less than or equal to the Upper Desired Temperature, the value of
  lthe Heat Control shall not be changed.":
  assume (regulator_mode == Isolette_Data_Model::Regulator_Mode.Normal_Regulator_Mode)
    & (current_tempWstatus.value >= lower_desired_temp.value
    & current_tempWstatus.value <= upper_desired_temp.value);
  guarantee heat_control == In(lastCmd);

case REQ_MHS_5 "If the Regulator Mode is FAILED, the Heat Control shall be
  lset to Off.":
  assume regulator_mode == Isolette_Data_Model::Regulator_Mode.Failed_Regulator_Mode;
  guarantee heat_control == Isolette_Data_Model::On_Off.Off;
```

DOT/FAA/AR-08/32
Air Traffic Organization
Research & Operations Planning
Office of Research and
Technology Development
Washington, DC 20581

Requirements Engineering
Management Handbook

REQ-MHS-1: If the Regulator Mode is INIT, the Heat Control shall be set to Off.
Rationale: A regulator that is initializing cannot regulate the Current Temperature of the Isolette and the Heat Control should be turned off.

REQ-MHS-2: If the Regulator Mode is NORMAL and the Current Temperature is less than the Lower Desired Temperature, the Heat Control shall be set to On.

REQ-MHS-3: If the Regulator Mode is NORMAL and the Current Temperature is greater than the Upper Desired Temperature, the Heat Control shall be set to Off.

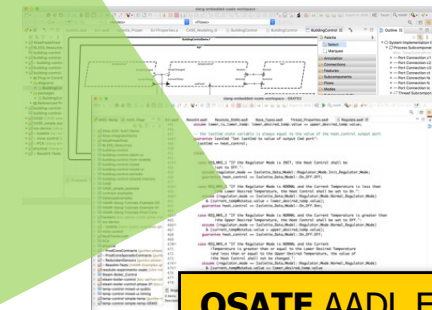
REQ-MHS-4: If the Regulator Mode is NORMAL and the Current Temperature is greater than or equal to the Lower Desired Temperature and less than or equal to the Upper Desired Temperature, the value of the Heat Control shall not be changed.

REQ-MHS-5: If the Regulator Mode is FAILED, the Heat Control shall be set to Off.

U.S. Department of Transportation
Federal Aviation Administration



*Developer
formalizes
requirements*



OSATE AADL Editor

Manage Heat Source Contracts

AADL GUMBO Contracts for **Manage Heat Source** Thread, with traceability to REMH requirements.

Mode condition

Compare current temperature to desired range

...

```
case REQ_MHS_2 "If the Regulator Mode is NORMAL and the Current Temperature is less than  
the Lower Desired Temperature, the Heat Control shall be set to On.":  
  assume (regulator_mode == Isolette_Data_Model::Regulator_Mode.Normal_Regulator_Mode)  
    & (current_tempWstatus.value < lower_desired_temp.value);  
  guarantee heat_control == Isolette_Data_Model::On_Off.Onn;
```

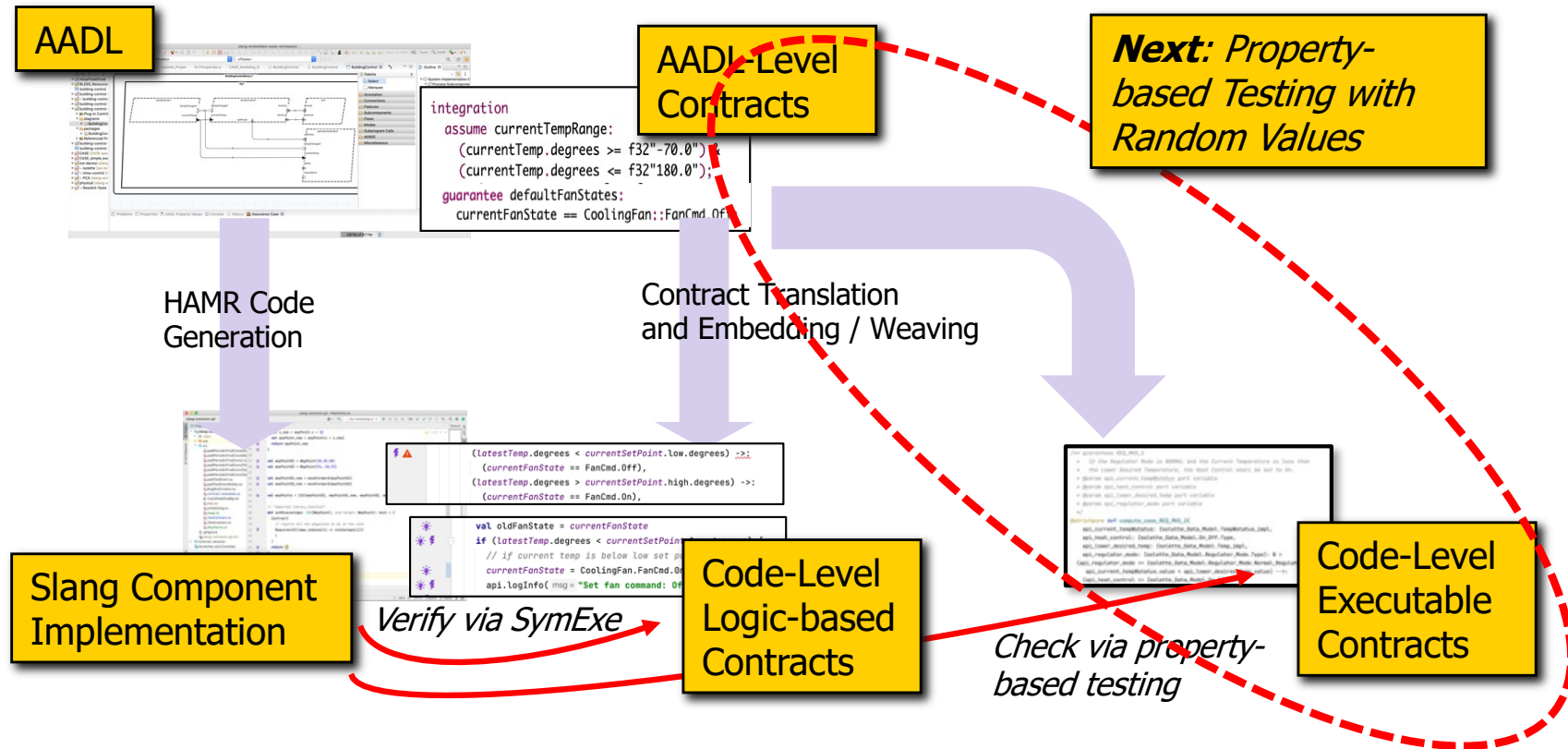
...

Set the desired state of the heater



Application: *Property-based (Contract-based) Testing*

KSU / Adventium Labs (now Galois) ARMY SBIR Phase II...



Manage Heat Source Contracts

Translation of **model-level** GUMBO contracts to Slang **code-level** executable contracts

AADL GUMBO Contract (clause)

```
case REQ_MHS_2 "If the Regulator Mode is NORMAL and the Current Temperature is less than  
the Lower Desired Temperature, the Heat Control shall be set to On."  
  assume (regulator_mode == Isolette_Data_Model::Regulator_Mode.Normal_Regulator_Mode)  
    & (current_tempWstatus.value < lower_desired_temp.value);  
  guarantee heat_control == Isolette_Data_Model::On_Off.Onn;
```

Model

auto-generated

ito-generated

Application
Code

Code

Library of
Executable
Contracts

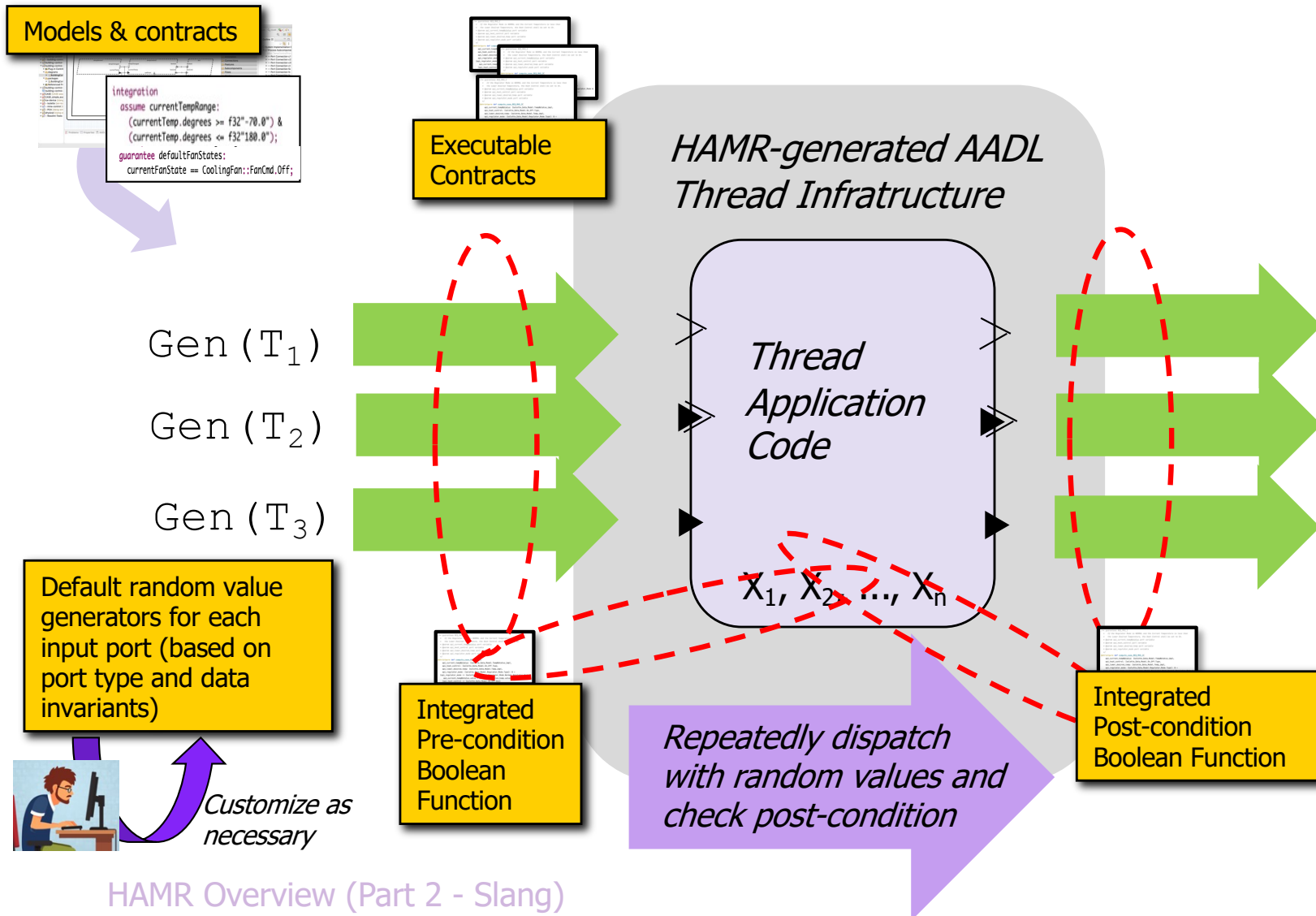
Slang Executable Contract (clause)

```
/** guarantees REQ_MHS_2  
 *   If the Regulator Mode is NORMAL and the Current Temperature is less than  
 *   the Lower Desired Temperature, the Heat Control shall be set to On.  
 * @param api_current_tempWstatus port variable  
 * @param api_heat_control port variable  
 * @param api_lower_desired_temp port variable  
 * @param api_regulator_mode port variable  
 */  
@strictpure def compute_case_REQ_MHS_2(  
  api_current_tempWstatus: Isolette_Data_Model.TempWstatus_impl,  
  api_heat_control: Isolette_Data_Model.On_Off.Type,  
  api_lower_desired_temp: Isolette_Data_Model.Temp_impl,  
  api_regulator_mode: Isolette_Data_Model.Regulator_Mode.Type): B =  
  (api_regulator_mode == Isolette_Data_Model.Regulator_Mode.Normal_Regulator_Mode &  
   api_current_tempWstatus.value < api_lower_desired_temp.value) -->:  
  (api_heat_control == Isolette_Data_Model.On_Off.Onn)
```

Traceability info automatically embedded

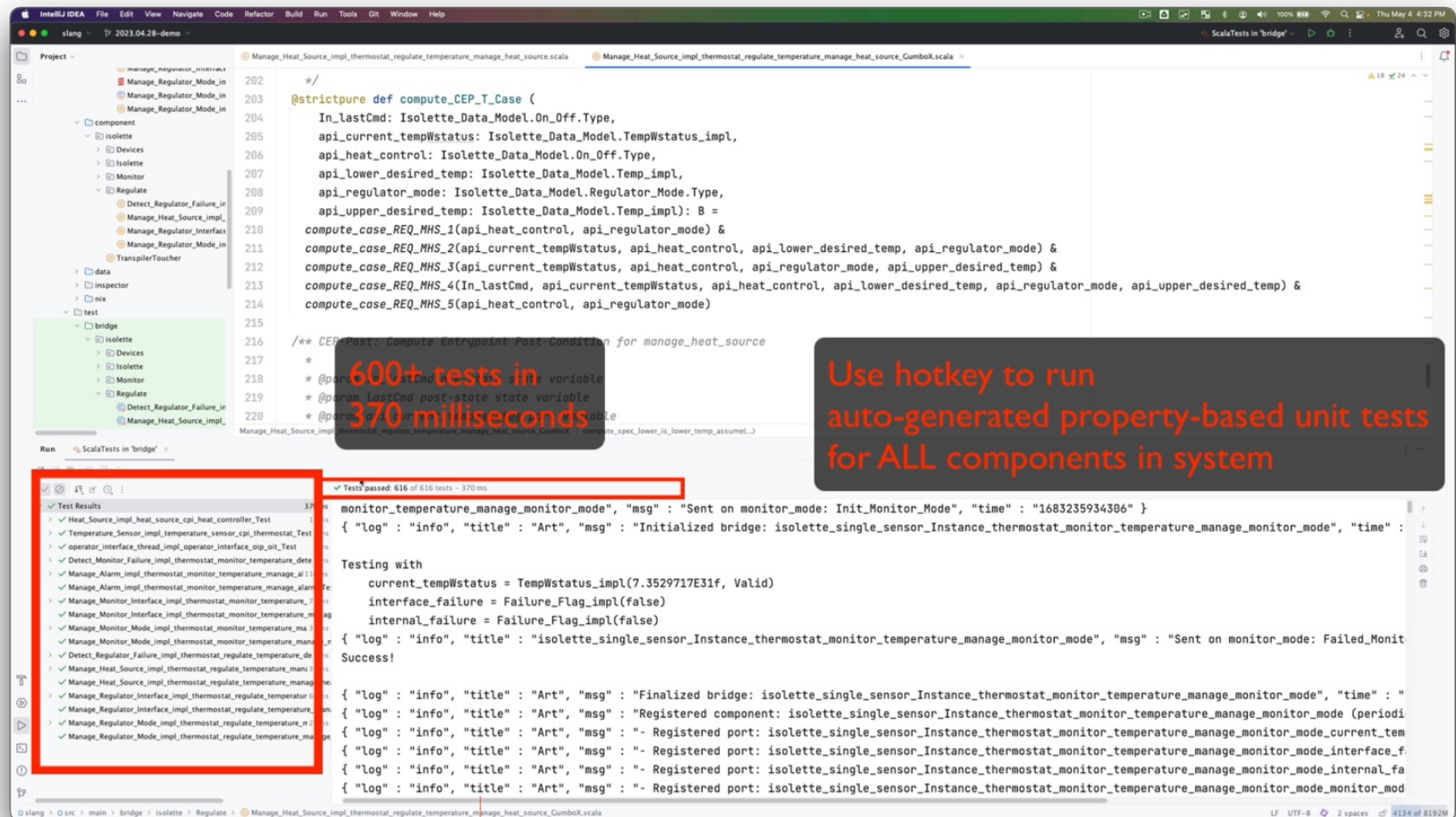
Auto-Generated Property-based Testing Harness

For every thread component, HAMR auto-generates property-based testing infrastructure for inserting values into component input ports and for checking values of output ports.



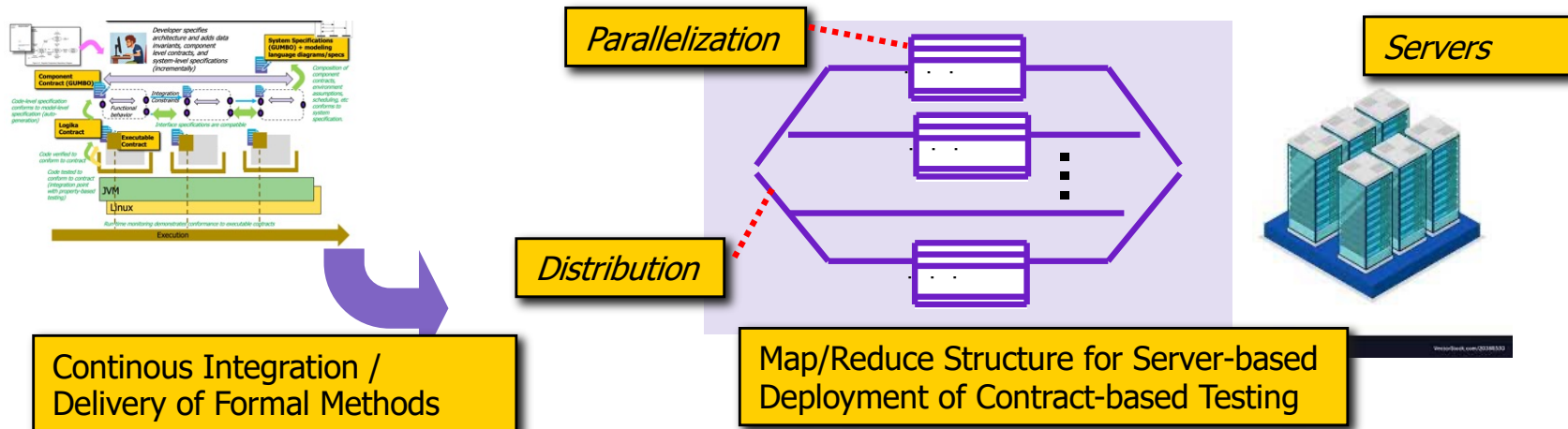
Demo

Property-based Testing



Scaling Up - Server-Based Deployment

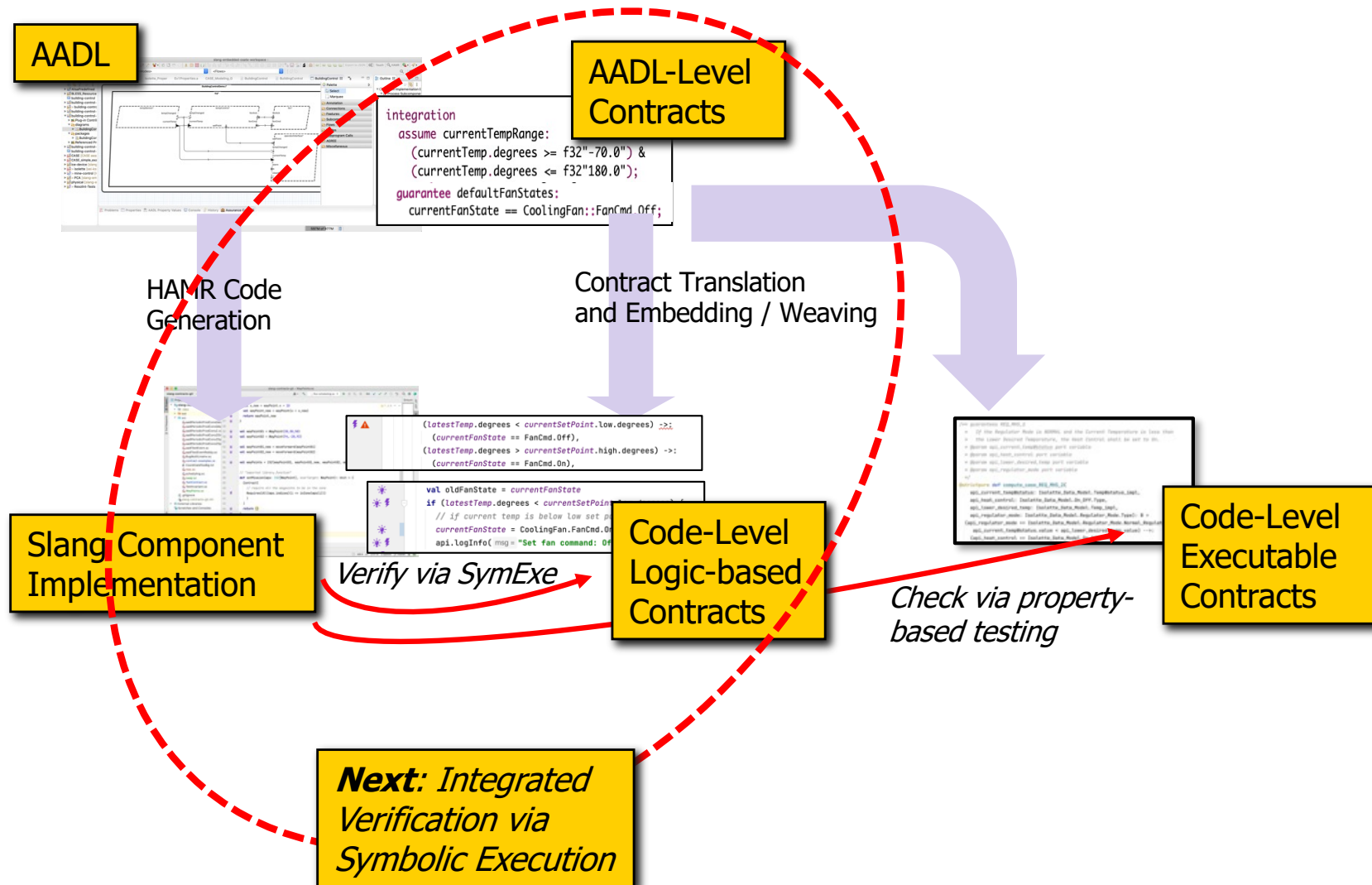
HAMR generates a server-based deployment to run the framework in a distributed/parallel fashion...



- Random generators and contract-based tests are farmed out to a configurable family of servers
- Test vectors and results are serialized for flexible deployment, reporting, and replay of the tests
- Currently hosted using our Jenkins setup, but easy for HAMR to automatically generate deployment scripts, e.g., for AWS, in the future

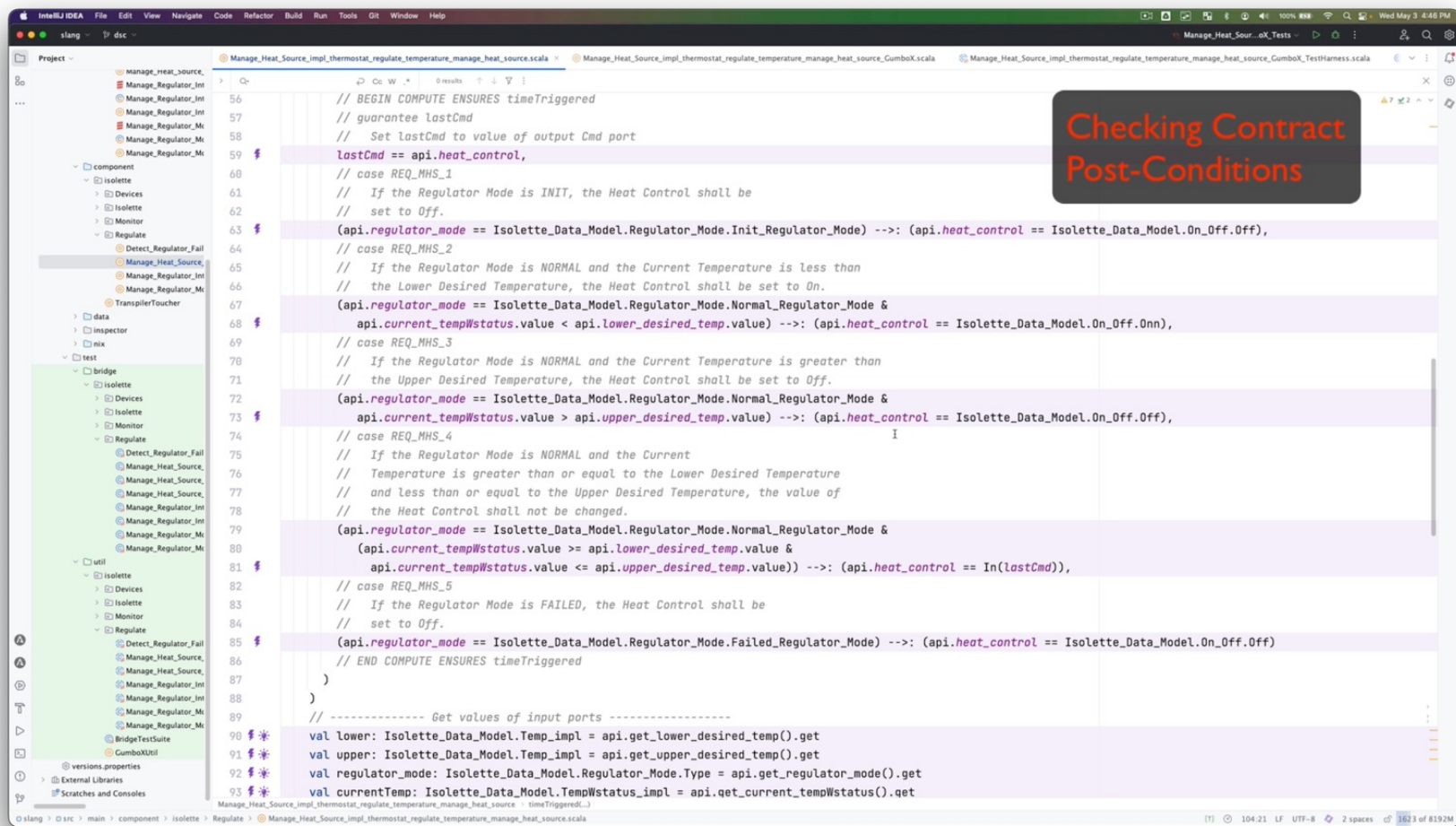
Application: *Usable, Workflow Integrated Verification*

KSU / Adventium Labs (now Galois) DARPA SBIR Phase II...



Demo

Verification against contracts using Symbolic Execution

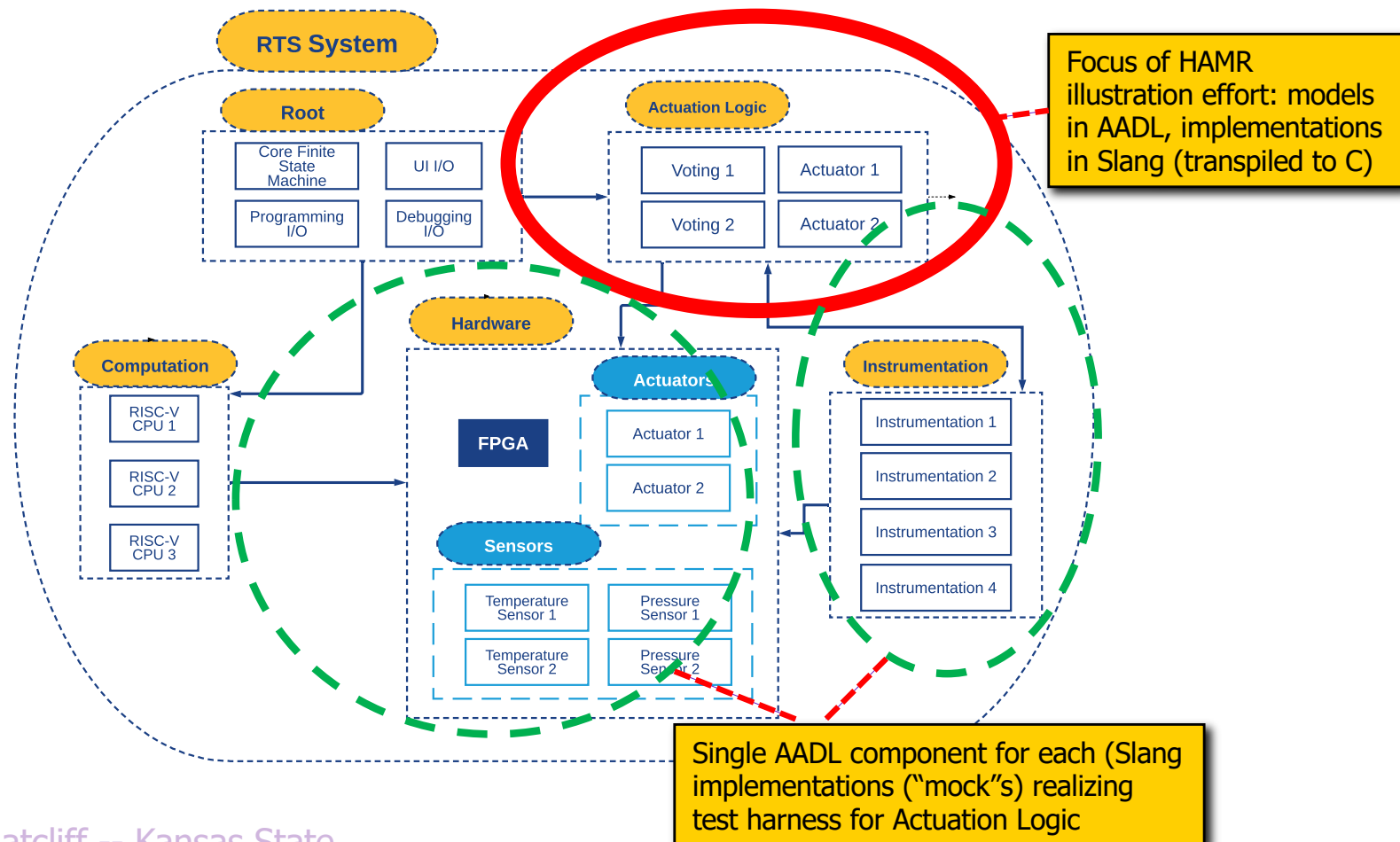


The screenshot displays the IntelliJ IDEA IDE with a Scala project. The left sidebar shows a project tree with folders like 'component', 'data', 'test', and 'util'. The main editor window shows a Scala file named 'Manage_Heat_Source_impl_thermostat_regulate_temperature_manage_heat_source.scala'. The code is a Scala function that implements a heat source regulator, featuring comments in English and Scala code in French. A callout box with the text 'Checking Contract Post-Conditions' is overlaid on the right side of the code. The code includes comments in English and Scala code in French, describing the logic for setting the heat control based on the regulator mode and current temperature. The code is as follows:

```
56 // BEGIN COMPUTE ENSURES timeTriggered
57 // guarantee lastCmd
58 // Set lastCmd to value of output Cmd port
59 lastCmd == api.heat_control,
60 // case REQ_MHS_1
61 // If the Regulator Mode is INIT, the Heat Control shall be
62 // set to Off.
63 (api.regulator_mode == Isolette_Data_Model.Regulator_Mode.Init_Regulator_Mode) -->: (api.heat_control == Isolette_Data_Model.On_Off.Off),
64 // case REQ_MHS_2
65 // If the Regulator Mode is NORMAL and the Current Temperature is less than
66 // the Lower Desired Temperature, the Heat Control shall be set to On.
67 (api.regulator_mode == Isolette_Data_Model.Regulator_Mode.Normal_Regulator_Mode &
68 api.current_tempWstatus.value < api.lower_desired_temp.value) -->: (api.heat_control == Isolette_Data_Model.On_Off.Onn),
69 // case REQ_MHS_3
70 // If the Regulator Mode is NORMAL and the Current Temperature is greater than
71 // the Upper Desired Temperature, the Heat Control shall be set to Off.
72 (api.regulator_mode == Isolette_Data_Model.Regulator_Mode.Normal_Regulator_Mode &
73 api.current_tempWstatus.value > api.upper_desired_temp.value) -->: (api.heat_control == Isolette_Data_Model.On_Off.Off),
74 // case REQ_MHS_4
75 // If the Regulator Mode is NORMAL and the Current
76 // Temperature is greater than or equal to the Lower Desired Temperature
77 // and less than or equal to the Upper Desired Temperature, the value of
78 // the Heat Control shall not be changed.
79 (api.regulator_mode == Isolette_Data_Model.Regulator_Mode.Normal_Regulator_Mode &
80 (api.current_tempWstatus.value >= api.lower_desired_temp.value &
81 api.current_tempWstatus.value <= api.upper_desired_temp.value)) -->: (api.heat_control == In(lastCmd)),
82 // case REQ_MHS_5
83 // If the Regulator Mode is FAILED, the Heat Control shall be
84 // set to Off.
85 (api.regulator_mode == Isolette_Data_Model.Regulator_Mode.Failed_Regulator_Mode) -->: (api.heat_control == Isolette_Data_Model.On_Off.Off)
86 // END COMPUTE ENSURES timeTriggered
87 )
88 )
89 // ----- Get values of input ports -----
90 val lower: Isolette_Data_Model.Temp_impl = api.get_lower_desired_temp().get
91 val upper: Isolette_Data_Model.Temp_impl = api.get_upper_desired_temp().get
92 val regulator_mode: Isolette_Data_Model.Regulator_Mode.Type = api.get_regulator_mode().get
93 val currentTemp: Isolette_Data_Model.TempWstatus_impl = api.get_current_tempWstatus().get
```

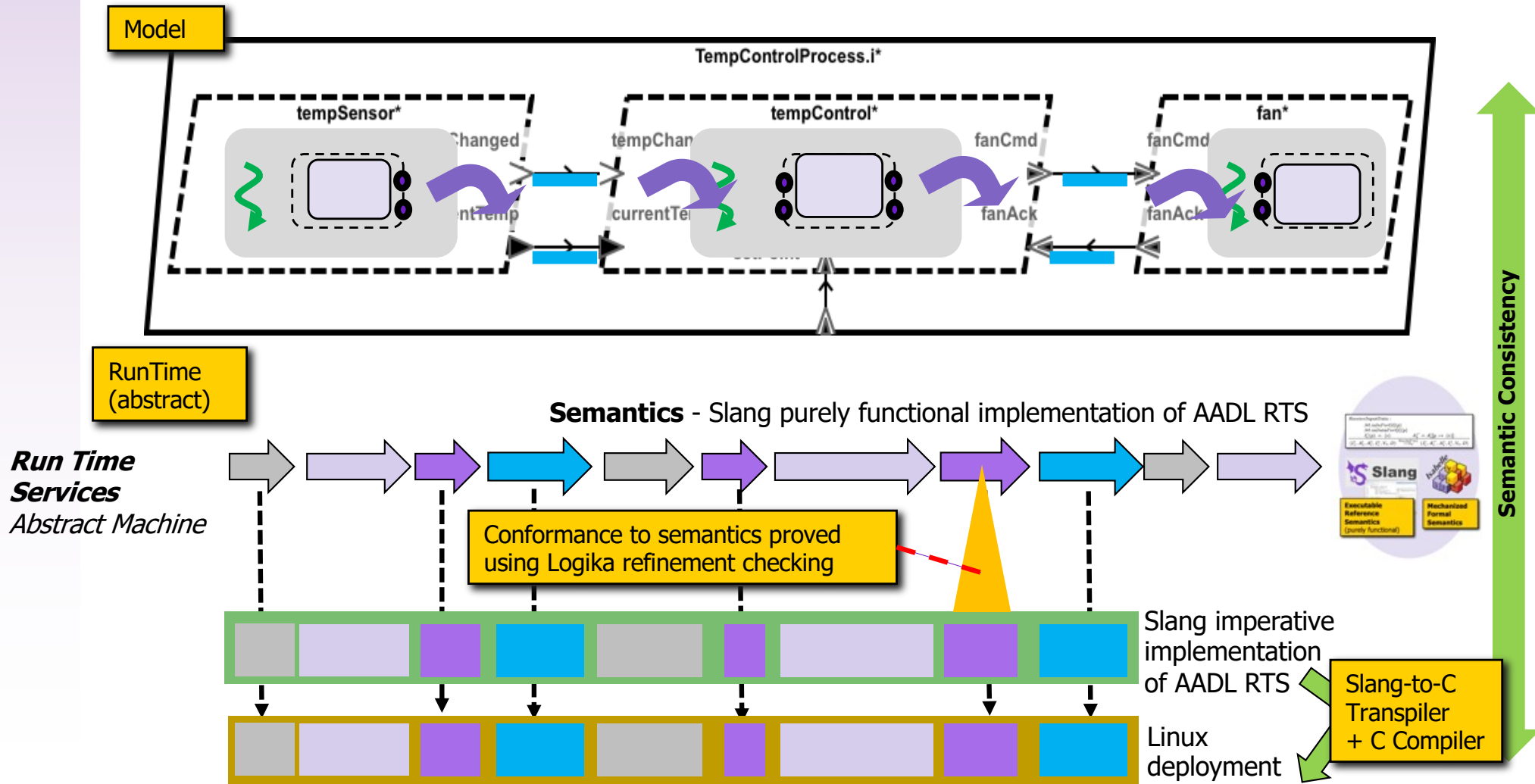
Galois / NRC RTS Example

This methodology has also been applied end-to-end (down to JVM, Linux, seL4) for Galois' open source implementation of a **nuclear reactor trip system** (US NRC funded work) – *available end of May*



Verifying Deployed Infrastructure Code

Ongoing work: Using Logika refinement checking to verify that deployed infrastructure satisfies the semantics...



Automatically Verifying AADL RTS Implementation

Using Logika refinement checking, we can automatically verify that Slang-based implementation of AADL run-time services conforms to Slang purely functional reference semantics

Verification of `ReceiveInput` service implementation

hamr-exec-semantics-idea > hamr-exec-semantics > src > main > scala > aadlsem > rts-refinement-simplified.sc

Project: hamr-exec-semantics-idea

Structure:

- hamr-exec-semantics-idea
 - .idea
 - bin
 - hamr-exec-semantics
 - src
 - main
 - scala
 - aadlsem
 - AppCode
 - Comm
 - Contracts
 - DispatchLogic
- test
 - out
 - .gitattributes
 - .gitignore
 - External Libraries
 - Scratches and Consoles

Code:

```
@pure def receiveInputOne(isEvent: B, kind: Semantics.PortProperties.Kind, inInfraSemPre: Semantics.PortContents, inInfraSemPost: Semantics.PortContents, inInfraImplPre: Impl.PortContents, inInfraImplPost: Impl.PortContents): B = {  
  Contract(  
    Requires(  
      Semantics.Spec.inv(inInfraSemPre, inAppSemPre, numOfPortIds),  
      Impl.inv(inInfraImplPre, inAppImplPre, numOfPortIds),  
      inInfraSemPre.contains(portId),  
      !isEvent -> !inInfraImplPre(portId).isInstanceOf[DataContent.NoData],  
      isEvent == (kind == Semantics.PortProperties.Kind.Event),  
      portStatesContents(inInfraSemPre, inAppSemPre, inInfraImplPre, inAppImplPre, isEvent, portId),  
      Semantics.Spec.ReceiveInput.behavior(inInfraSemPre, inAppSemPre, kind, portId, inInfraSemPost, inAppSemPost),  
      Impl.Spec.ReceiveInput.behavior(inInfraImplPre, inAppImplPre, isEvent, portId, inInfraImplPost, inAppImplPost)  
    ),  
    Ensures(  
      portStatesContents(inInfraSemPost, inAppSemPost, inInfraImplPost, inAppImplPost, isEvent, portId)  
    )  
  )  
}
```

Annotations:

- Run the spec...
- Run the implementation...
- Prove refinement relation holds between spec and implementation post-state.
- Assume well-formedness conditions on both spec and implementation state.
- Assume refinement relation holds between spec and implementation pre-state.

Logika Verified
Programming logic proof is accepted

273:8 LF UTF-8 2 spaces main

Conclusion

HCSS Theme: *"Semantically Rigorous and Integrated High-Level Abstractions"*

- Foundations provided by a formal / mechanized semantics of AADL Run-Time Services
 - Semantics artifacts will be in the HAMR release in mid-summer
 - Next steps: adding timing information, code-generation verification with connections to seL4 proofs
- Developer-friendly tool integrated contract framework for both testing and verification
 - Already available in HAMR distribution
- Continuing work on system-level verification and testing
- Looking for funding to develop additional HAMR backends for DoD platforms of interest