Protecting Safety-critical Software Intensive Systems From Malicious Action:

Strong partitioning using the verified seL4 microkernel and formal methods-integrated model-based development with SysMLv2 and Rust

SCC 2025 – May 16, 2025

Kansas State University

John Hatcliff
Robby
Jason Belt

Collins Aerospace DARPA PROVERS INSPECTA

- *Aarhus University*
- *CMU*
- *ProofCraft*
- *UNSW*

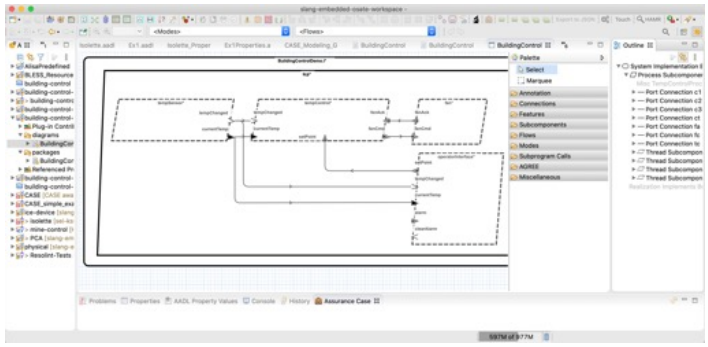
Galois

- *Todd Carpenter, Danielle Stewart*

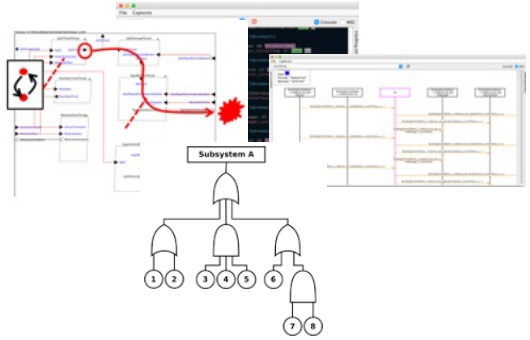
HAMR

HAMR – tool chain for [H]igh [A]ssurance [M]odeling and [R]apid engineering for embedded systems

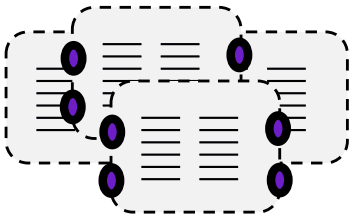
Modeling, analysis, and verification in the **AADL** modeling language (+ **SysMLv2**)



Leveraging analyses from AADL community



Component development and verification in multiple languages



- Slang (with contracts)
  - high integrity subset of Scala
  - contract verification framework
  - translates to C
  - translates to Rust
- Rust (with contracts)
- C

Deployments aligned with AADL run-time on multiple platforms

JVM Deployment

Linux Deployment

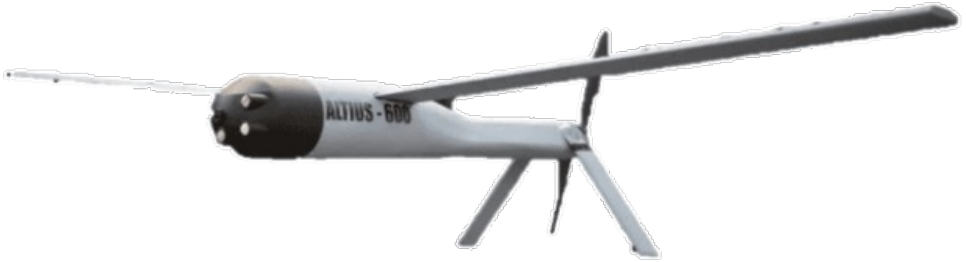
seL4 Deployment

HAMR – Collins Aerospace

On DARPA PROVERS, HAMR is being used to develop an experimental version of the infrastructure for Collins "Air Launched Effects" platform that provides increased modularity and security --- (final development will be HAMR SysMLv2 to Rust)

Current - Mission computer for UAVs collaborative UAVs

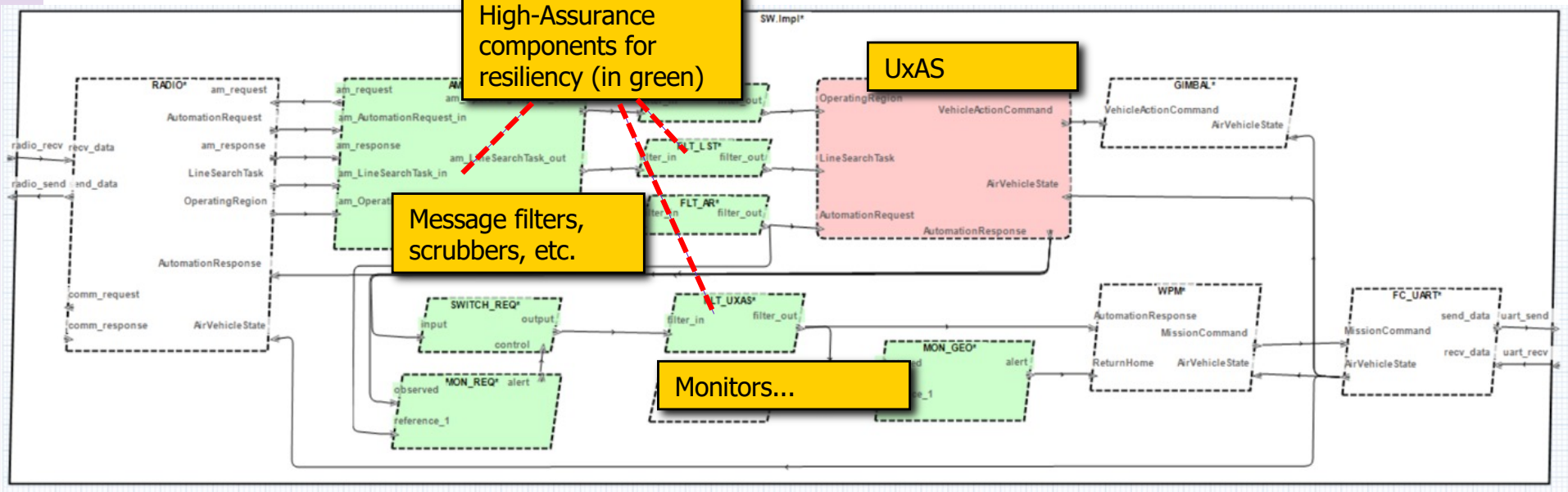
Video: https://youtu.be/SwPJHmZQMaM?si=NwTdb3VFpV-MxSre



Collins Aerospace

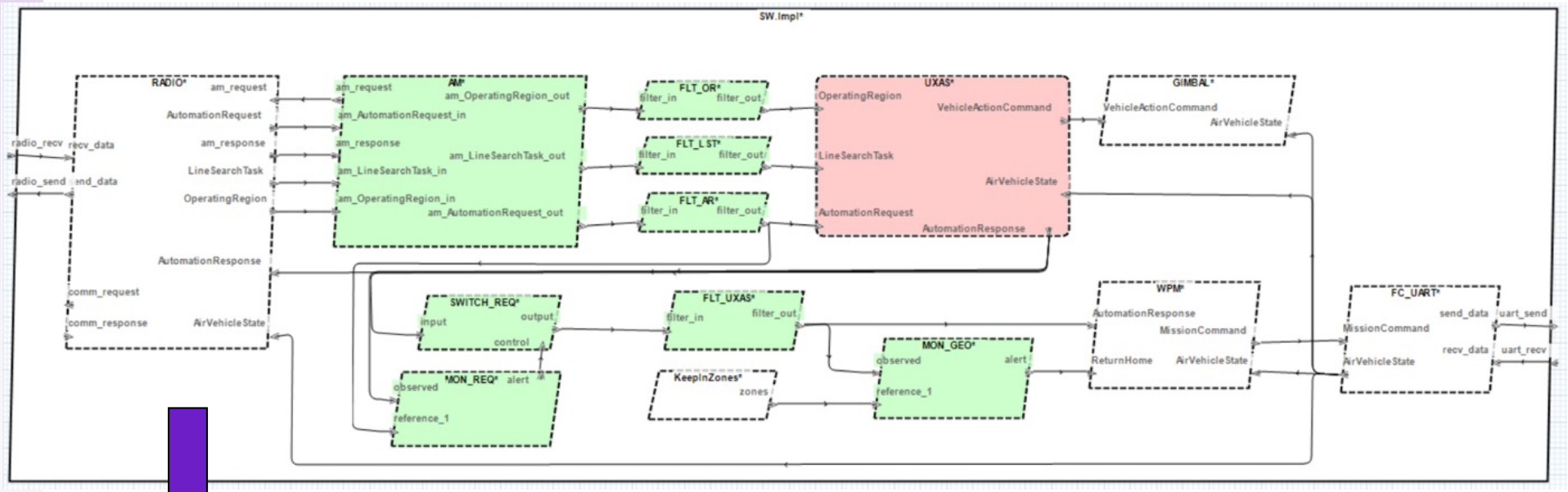
Separation Kernel Foundation - Verified Partitioning

Previous HAMR / Collins Aerospace demonstration based on AFRL UxAS – unmanned air system services – adding cyber-resiliency…



Separation Kernel Foundation - Verified Partitioning

Previous HAMR / Collins Aerospace demonstration based on AFRL UxAS – unmanned air system services



HAMR Code Generation for seL4 guarantees that **deployed system has the partitioning and information flow properties reflected in the model**

No infiltration

Cyber-resiliency (high criticality)

No exfiltration

Legacy code hosted **in VM** (low trust)

Fault containment

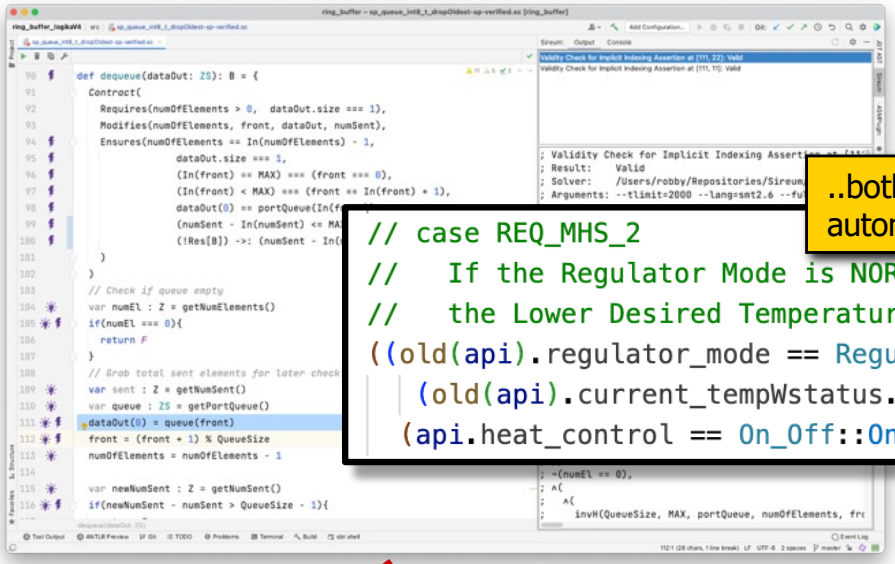
No eavesdropping; no interference

seL4 configuration

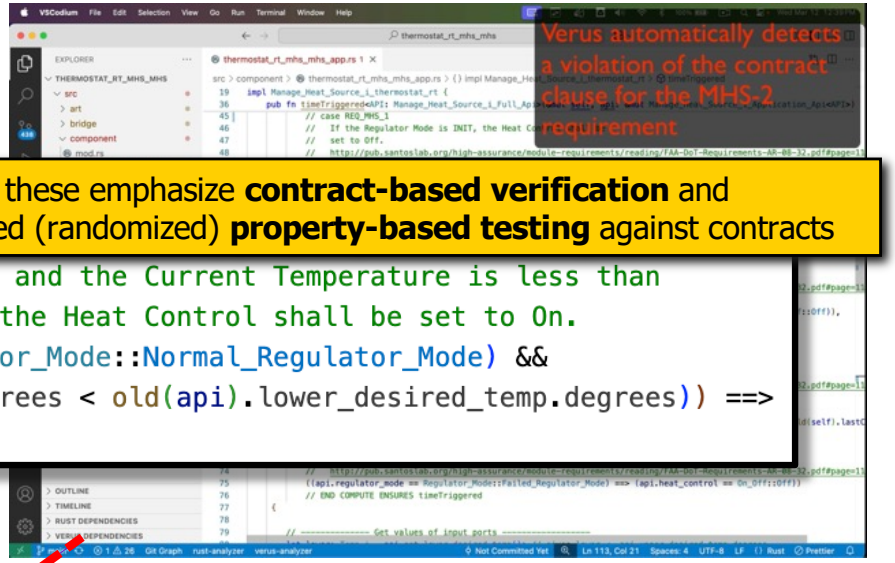
seL4 microkernel guarantees partitioning of components and communication (backed by computer-checked proofs)

Memory Safe Languages

Slang/Scala (with Logika verification)



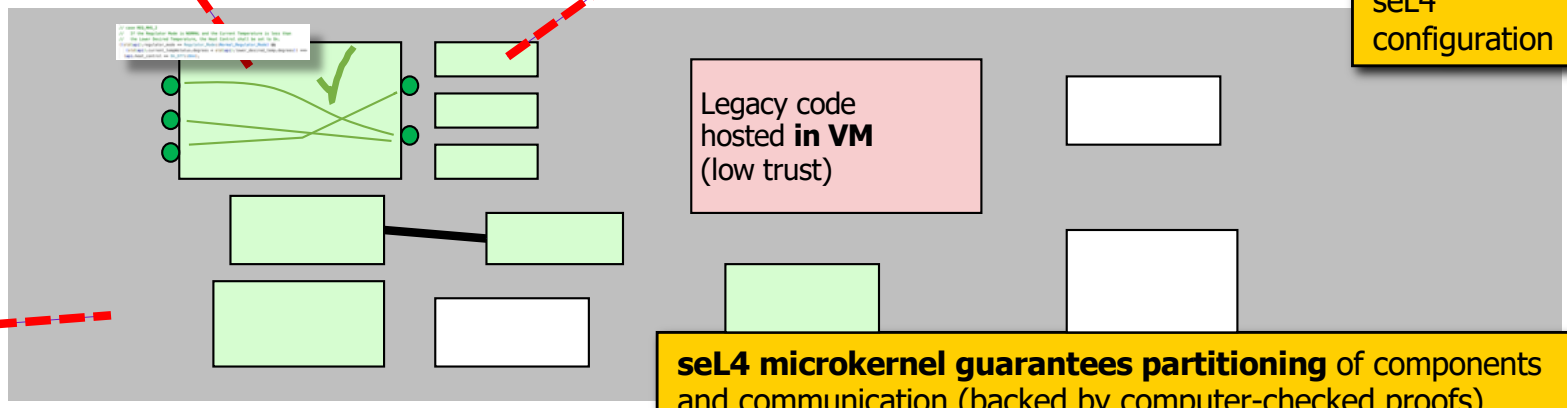
Rust (with Verus verification)



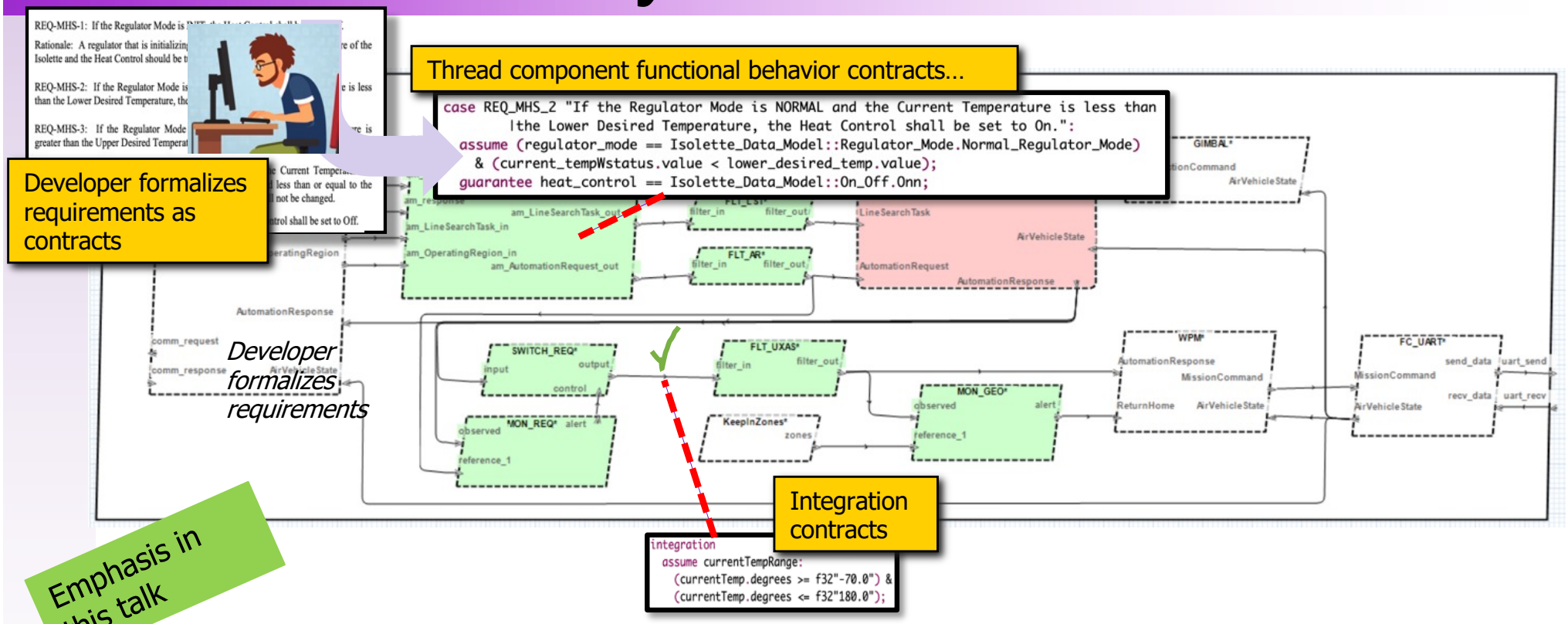

```
// case REQ_MHS_2
//   If the Regulator Mode is NORMAL and the Current Temperature is less than
//   the Lower Desired Temperature, the Heat Control shall be set to On.
((old(api).regulator_mode == Regulator_Mode::Normal_Regulator_Mode) &&
   (old(api).current_tempWstatus.degrees < old(api).lower_desired_temp.degrees)) ==>
  (api.heat_control == On_Off::Onn),
```

..both of these emphasize **contract-based verification** and automated (randomized) **property-based testing** against contracts

..maybe adding Frama-C to existing HAMR C code generation



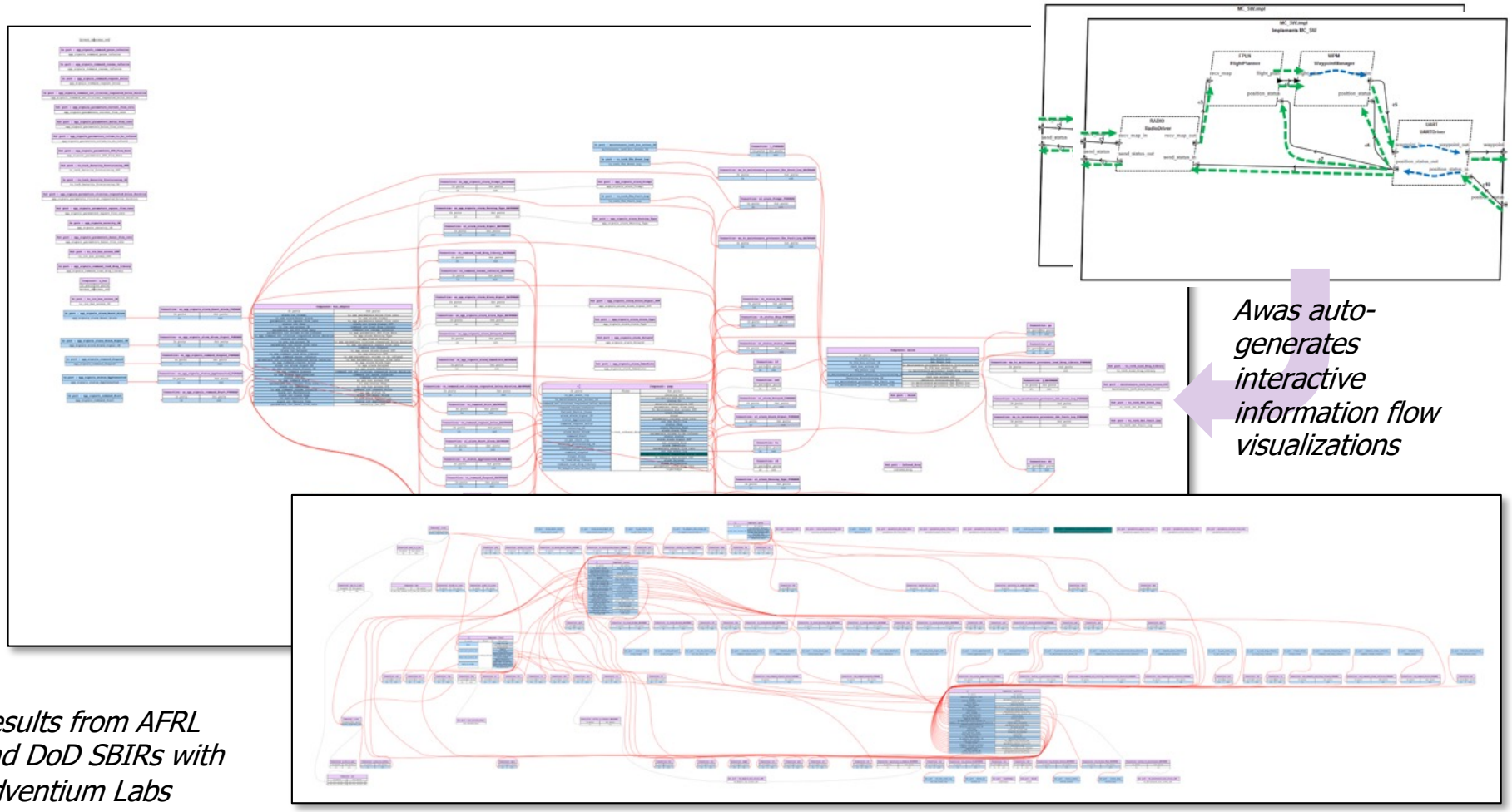
Models - Analyzeable Abstractions



- GUMBO model level contracts for thread components and component connections
- Integration contracts are checked in the SysMLv2 IDE
- Thread component contracts are transformed and embedded in generated application code skeletons for testing and verification

AADL Analysis Example: Information Flow

The KSU Awas tool (https://awas.sireum.org) generates scalable interactive visualizations of AADL information flows and model-based hazard analysis results



Awas auto-generates interactive information flow visualizations

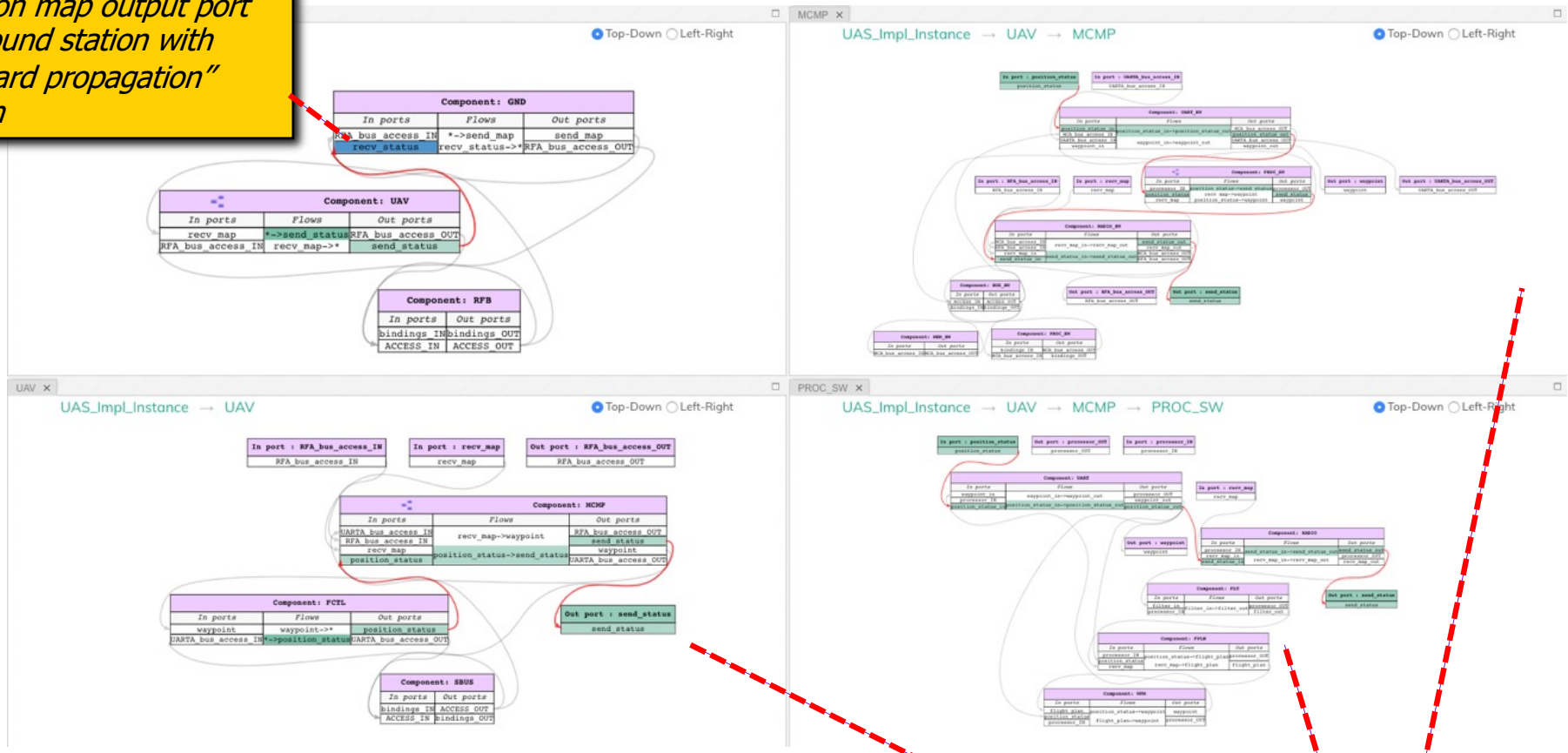
Results from AFRL and DoD SBIRs with Adventium Labs

Information flow graphs can be dynamically browsed and queried with path logic.

Interactive Browsing of Information Flows (AADL Level)

Example: In Ground Station / UAV example used on DARPA CASE, ask "How does map information propagate from ground station to UAV and through UAV's mission computer to produce a waypoint?"

Click on map output port of ground station with "forward propagation" option

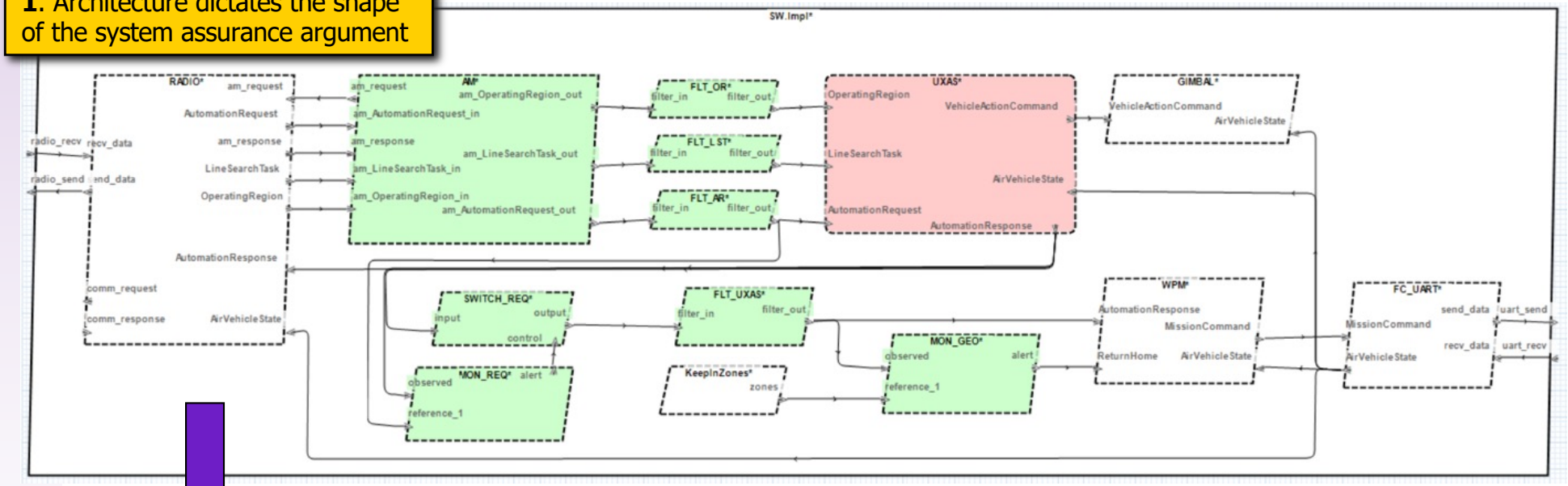


Immediately see results across different subsystems.

Separation Kernel Foundation - Verified Partitioning

Previous HAMR / Collins Aerospace demonstration based on AFRL UxAS – unmanned air system services

1. Architecture dictates the shape of the system assurance argument



HAMR Code Generation for seL4 guarantees that **deployed system has the partitioning and information flow properties reflected in the model**

Cyber-resiliency (high criticality)

Legacy code hosted **in VM** (low trust)

2. seL4 provides **assurance of the platform** and guarantees the **partitioning** and **information flow**.

3. Proven correct components establish assurance along specified information flow paths.

…assurance story goes back to Rushby's motivation for separation kernel

Isolette - Infant Incubator System

From US FAA Requirements Engineering Management Handbook (REMH)

- Handbook on best practices for requirements development written for the FAA by engineers at Collins Aerospace
- Includes example of an "Isolette" (infant incubator)

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

REMH

DOT/FAA/AR-08/32

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

Requirements Engineering Management Handbook

- Isolette described in 26-page Appendix (natural language requirements, tables, informal designs)
- Used as the running example in 100-page presentation of best practices

June 2009

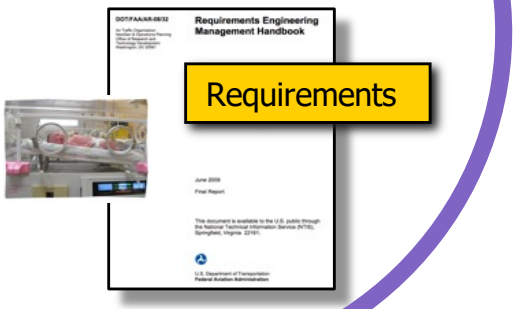
Final Report

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

https://www.faa.gov/sites/faa.gov/files/aircraft/air_cert/design_approvals/air_software/AR-08-32.pdf

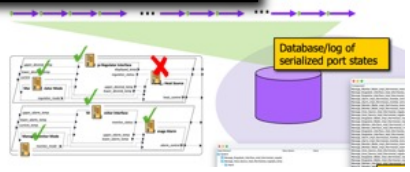
Much Effort to Make "End-to-End"



Requirements

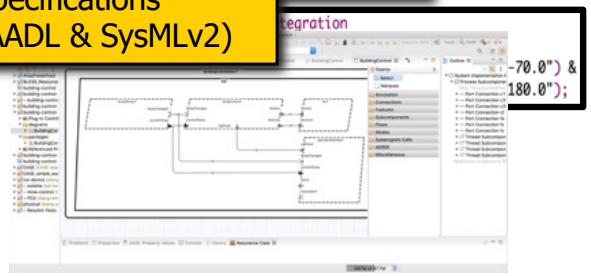
Concept of Operations

Run-time Monitoring

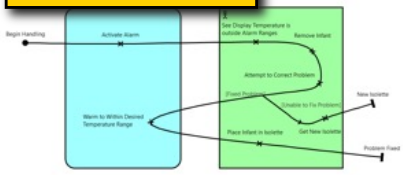


Architecture Specifications (AADL & SysMLv2)

+ Behavior ntracts

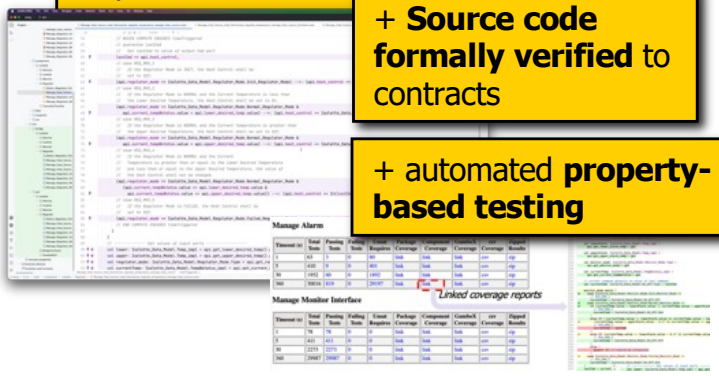


Semi-Formal and Formal Use Cases



Deployments on JVM for Simulation

Software Implementation

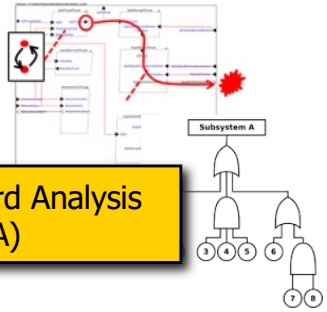


+ **Source code formally verified** to contracts

+ automated **property-based testing**

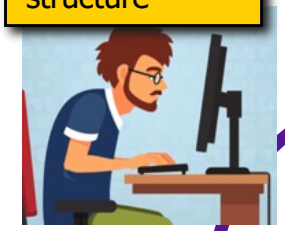
Deployments on seL4 verified micro-kernel

Hazard Analysis (STPA)



AADL Modeling Concepts

Developer configures computational structure



AADL Thread Property Options

Periodic
Sporadic
Hybrid
…

+ timing, scheduling constraints, etc.

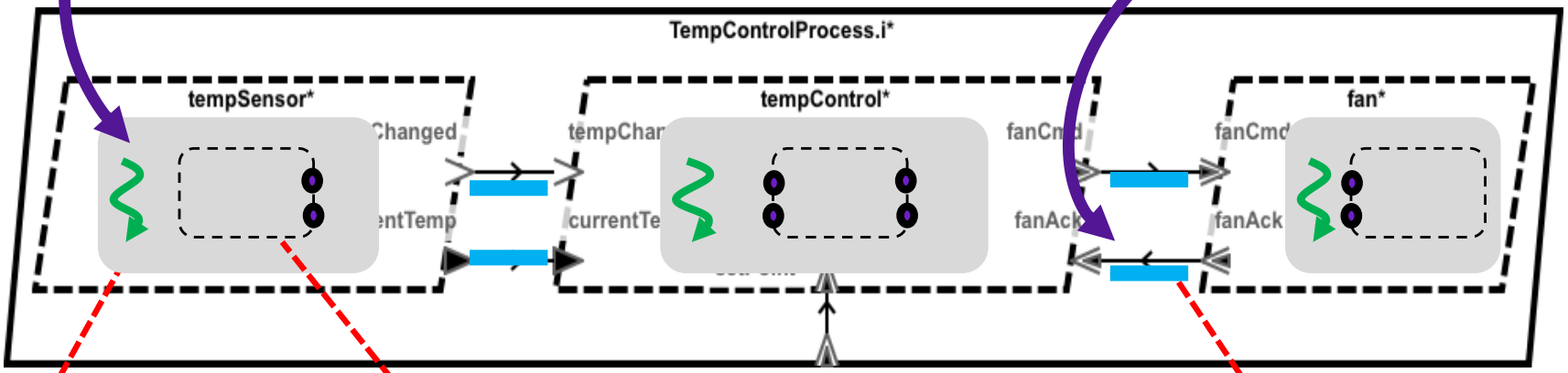
AADL Port & Connection Property Options

Event
Data

Event Data
…

buffered notifications
shared data cells
(or data distribution service)
buffered messages
(message passing middleware)

+ QoS, buffer sizes, latencies, etc



Code skeleton for selected thread pattern

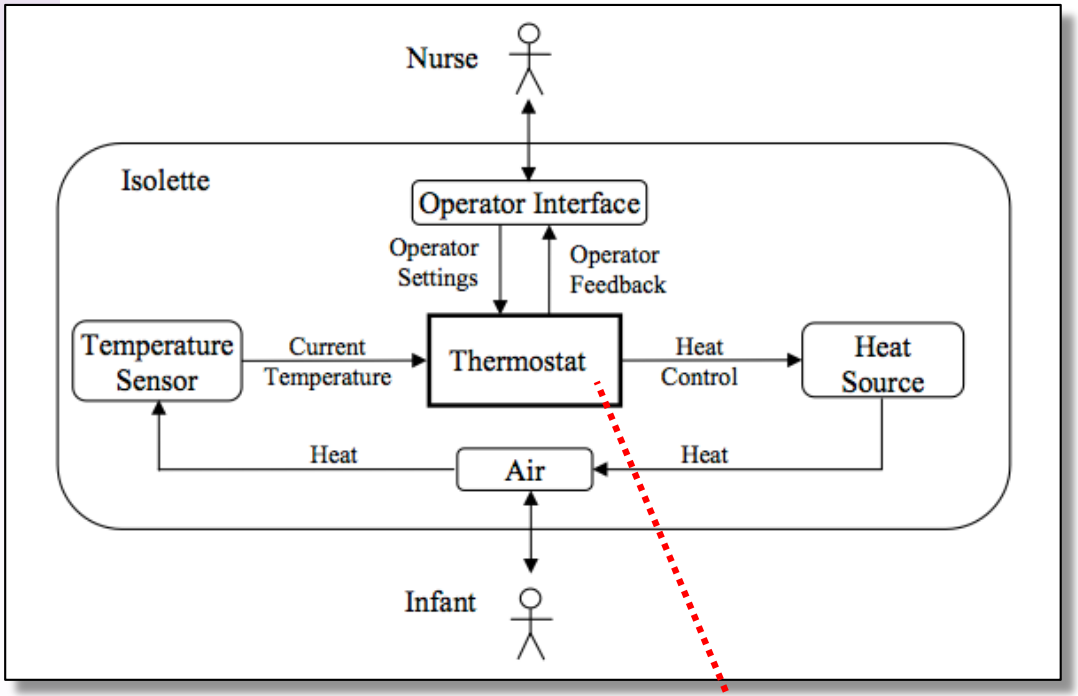
Implied API Pattern for application code to access port communication, etc.

Implementation of selected communication pattern

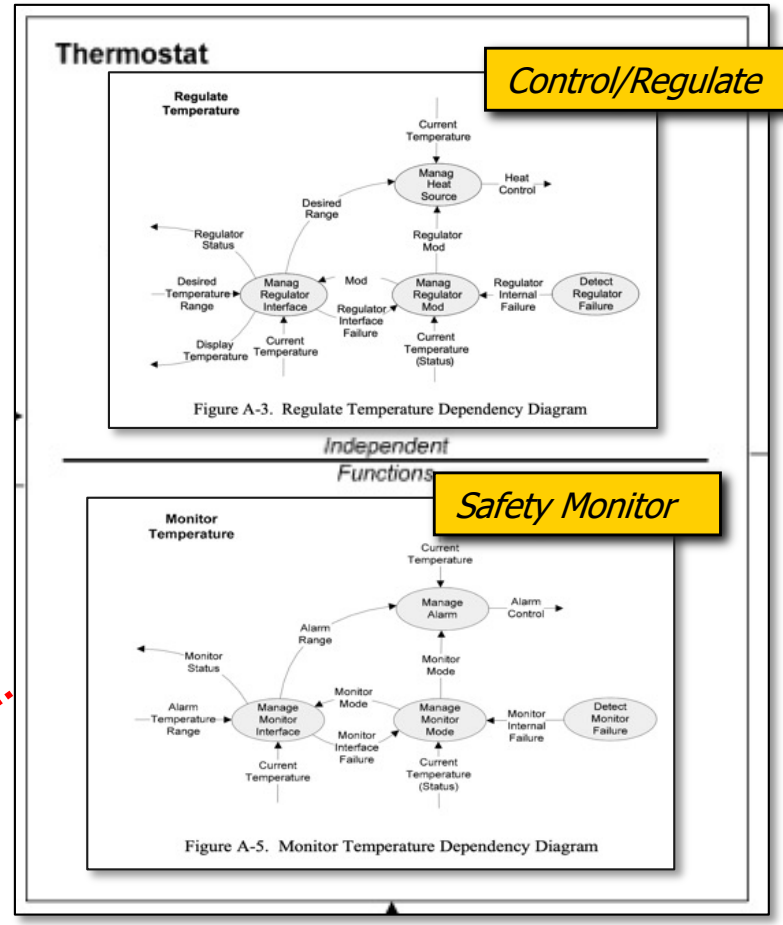
...an AADL contract language and testing infrastructure should be aligned with these patterns.

REMH - Informal Designs

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

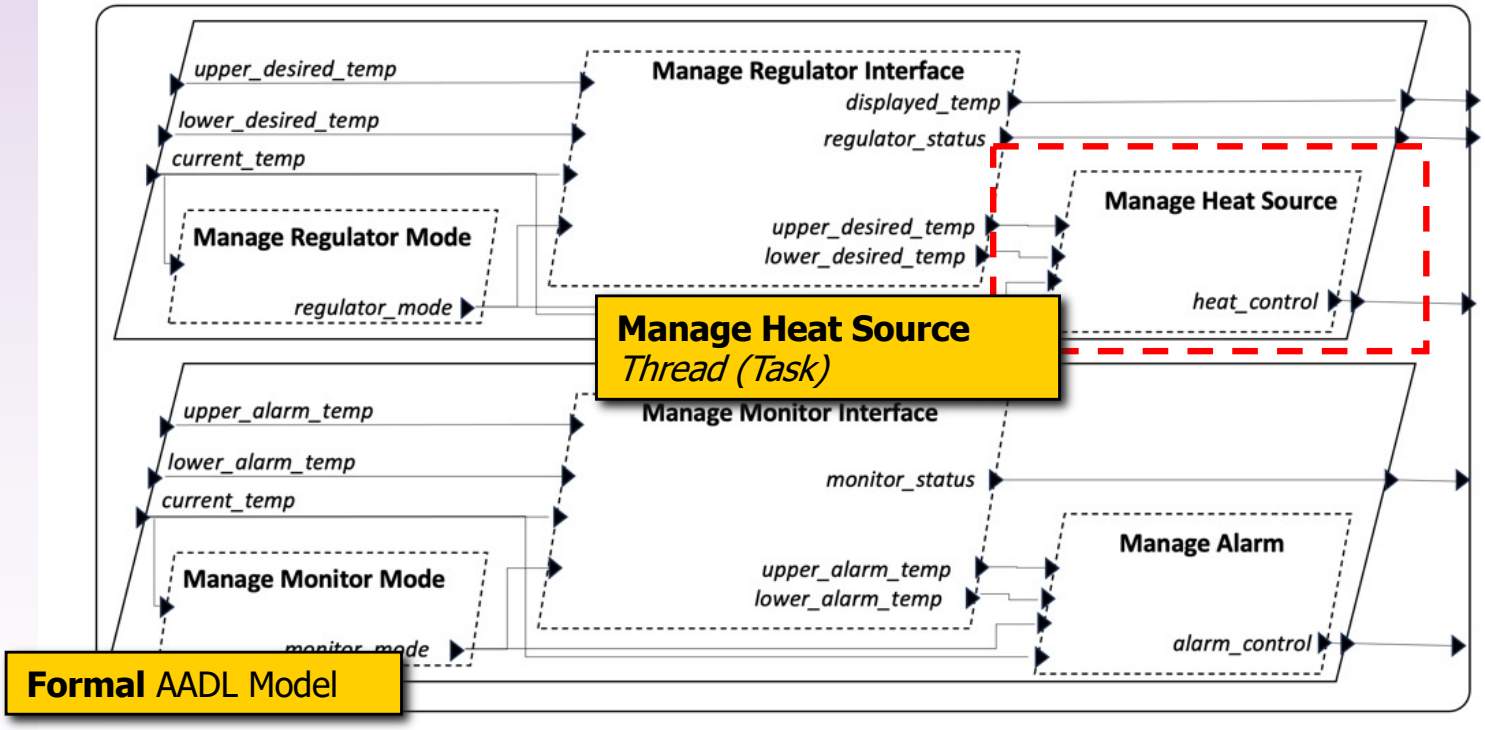


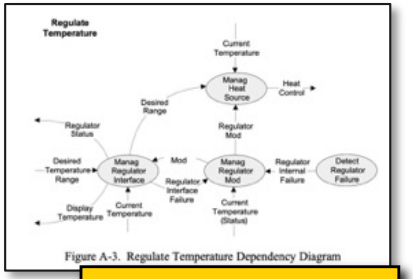
Thermostat decomposed into Regulate Temperature and Monitor Temperature functions.



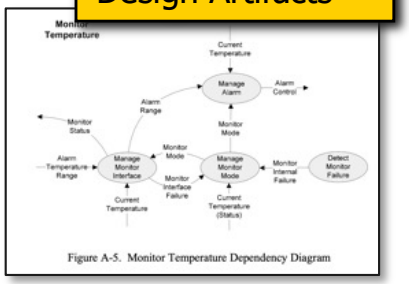
Using AADL to Represent Design

AADL Model is a straightforward rendering of the design diagrams in the FAA REMH





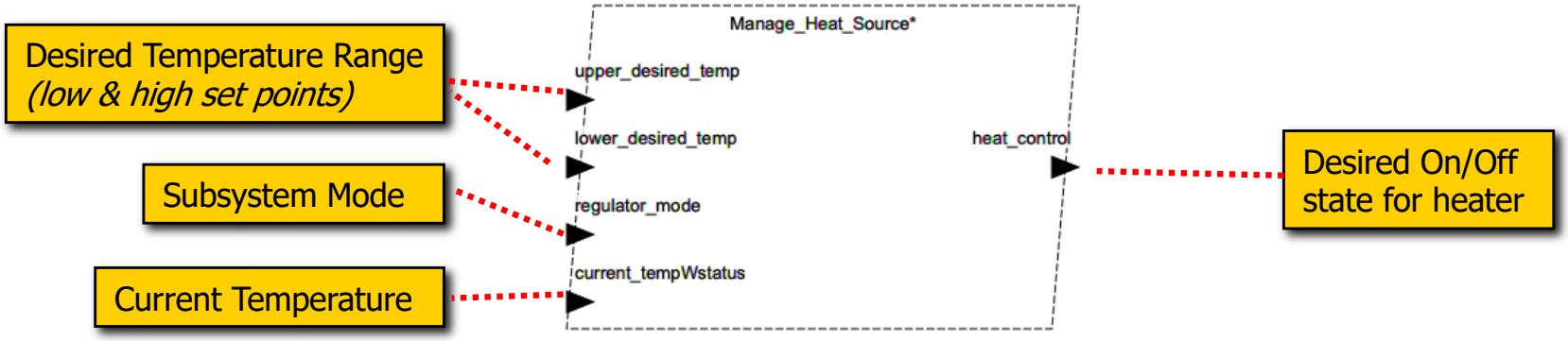
Informal REMH Design Artifacts



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

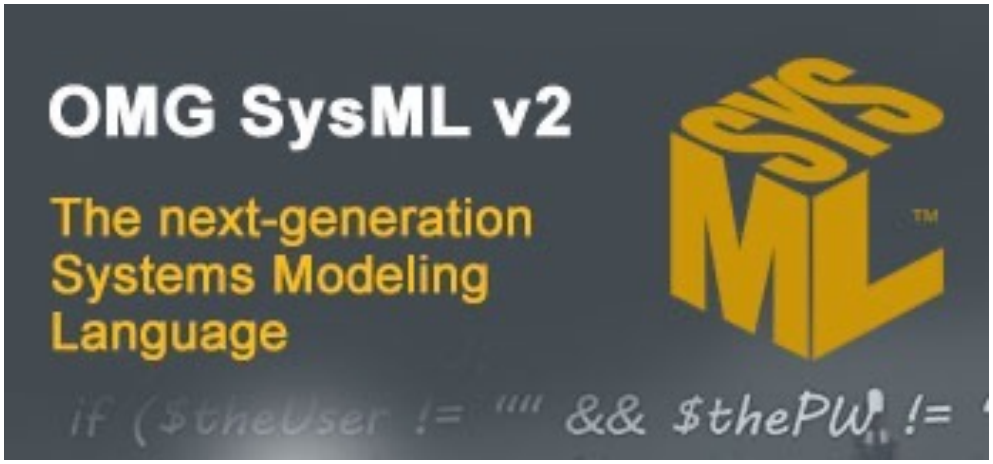
Manage Heat Source Thread

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;
```

SysMLv2



Why might SysMLv2 provide a alternate vehicle for rigorous model-based development, including AADL concepts?

- Will have wide-ranging commercial tool support as well as open source implementations
- Re-engineered from the ground up
  - No backwards compatibility with SysMLv1 except through translation
  - Not built as a profile of UML
- Like AADL, has both a graphical view and textual view
- Many AADL modeling elements have analogues in SysMLv2
  - E.g., components, ports, connections, developer-defined attributes
- Aims to provide a stronger "semantics" for system engineering compared to UML, SysMLv1

AADL / SysMLv2 Integration
OMG Standards Work

About the SMC

SMC

The OMG Systems Modeling Community gathers people interested in advancing SysMLv2

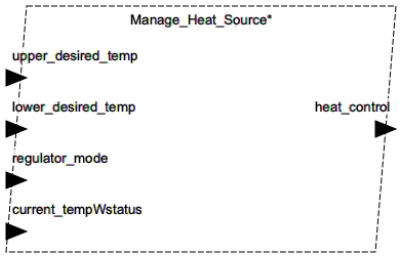
- Different membership structure
- See https://www.omg.org/communities/

RTESC Workgroup – entity responsible for integrating AADL concepts into SysMLv2

Charter: "*Develop domain libraries w/ KerML & SysMLv2 to support the precise modeling of Real-Time Embedded Safety-Critical Systems. Integrate capabilities from domain-specific models like SAE AADL, OpenGroup FACE, OMG MARTE, & AutoSAR*"

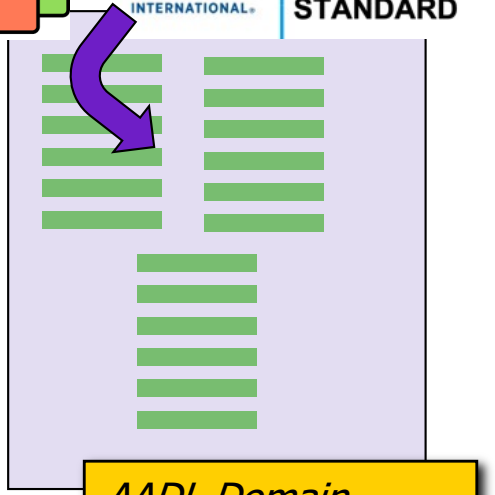
Lead: Gene Shreve (i3-Corp), Jerome Hugues (CMU/SEI)

Representing AADL in SysMLv2



RTESC workgroup represents AADL concepts as SysMLv2 types, attributes, etc.

SAE INTERNATIONAL AEROSPACE STANDARD



AADL Domain Library for SysMLv2

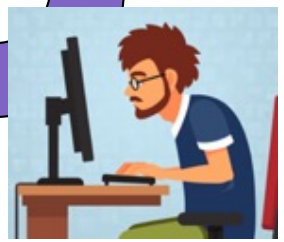
Mark as AADL ***thread***

SysMLv2

Set AADL port categories and types

```
part def Manage_Heat_Source_i :> Thread  {

  in port current_tempWstatus : DataPort { in :> type : Isolette_Data_Model::TempWstatus_i; }
  in port lower_desired_temp :  DataPort { in :> type : Isolette_Data_Model::Temp_i; }
  in port upper_desired_temp :  DataPort { in :> type : Isolette_Data_Model::Temp_i; }
  in port regulator_mode :  DataPort { in :> type : Isolette_Data_Model::Regulator_Mode; }

  out port heat_control : DataPort { out :> type : Isolette_Data_Model::On_Off; }

  attribute :>> Dispatch_Protocol = Supported_Dispatch_Protocols::Periodic;
  attribute :>> Period = 1000 [millisecond];
  attribute Domain: CASE_Scheduling::Domain = 9;
}
```

Set AADL pre-defined property values for this thread



Developer uses domain library to annotate base SysMLv2 elements with AADL concepts

AADL / SysMLv2 Component Types Visual Comparison

AADL

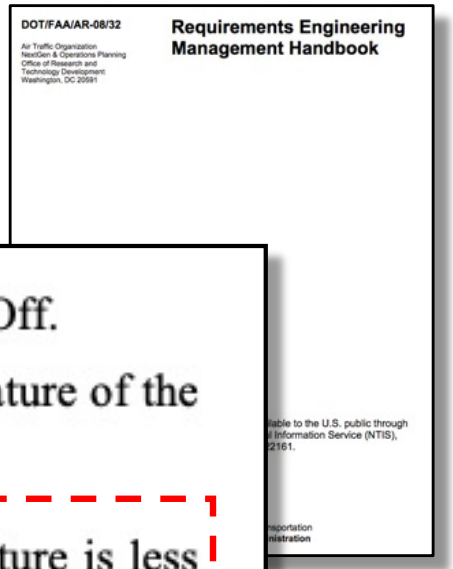
```
thread Manage_Heat_Source
 features
  current_tempWstatus: in data port Isolette_Data_Model::TempWstatus.impl;
  lower_desired_temp: in data port Isolette_Data_Model::Temp.impl;
  upper_desired_temp: in data port Isolette_Data_Model::Temp.impl;
  regulator_mode: in data port Isolette_Data_Model::Regulator_Mode;
  heat_control: out data port Isolette_Data_Model::On_Off;

 properties
  Dispatch_Protocol => Periodic;
  Period => Isolette_Properties::ThreadPeriod;
```

SysMLv2

```
part def Manage_Heat_Source_i :> Thread {

    in port current_tempWstatus : DataPort { in :> type : Isolette_Data_Model::TempWstatus_i; }
    in port lower_desired_temp : DataPort { in :> type : Isolette_Data_Model::Temp_i; }
    in port upper_desired_temp : DataPort { in :> type : Isolette_Data_Model::Temp_i; }
    in port regulator_mode : DataPort { in :> type : Isolette_Data_Model::Regulator_Mode; }
    out port heat_control : DataPort { out :> type : Isolette_Data_Model::On_Off; }

    attribute :>> Dispatch_Protocol = Supported_Dispatch_Protocols::Periodic;
    attribute :>> Period = 1000 [millisecond];
    attribute Domain: CASE_Scheduling::Domain = 9;
```

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.

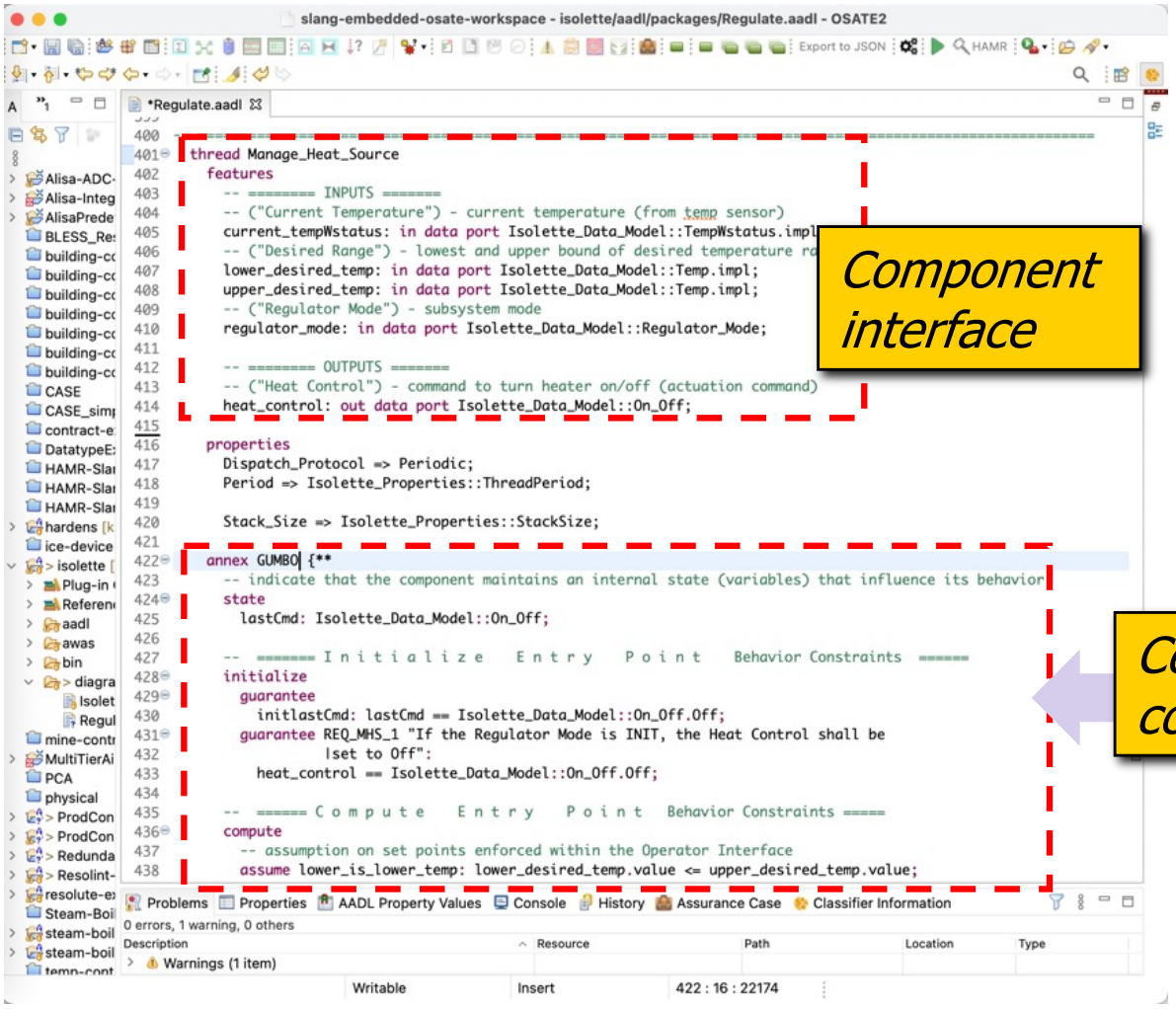
REQ-MHS-3: If the Regulator Mode is NORMAL and the Current Temperature is greater than the Upper Desired Temperature, the Heat Control shall be set to Off.

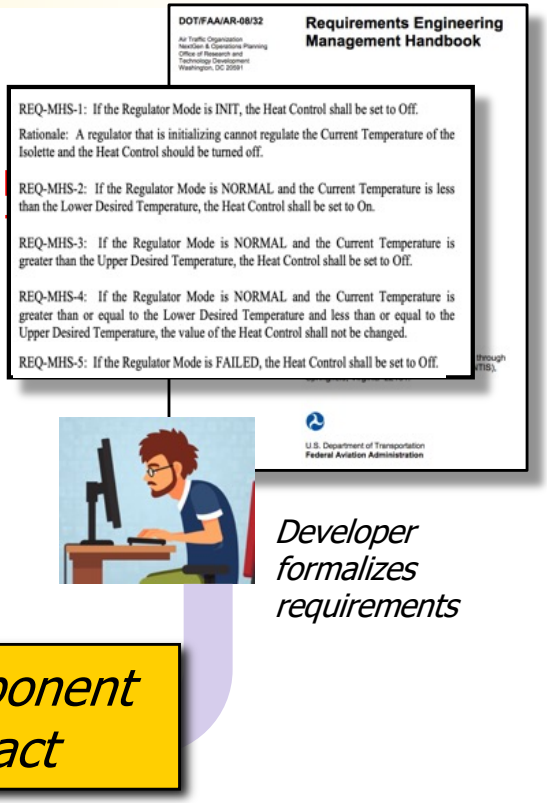
REQ-MHS-4: If the Regulator Mode is NORMAL and the Current Temperature is greater than or equal to the Lower Desired Temperature and less than or equal to the Upper Desired Temperature, the value of the Heat Control shall not be changed.

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

Requirements to Contracts

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



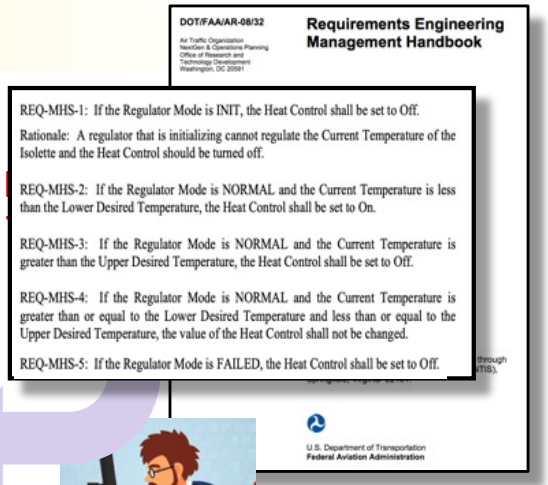


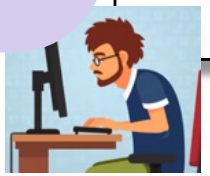
Developer formalizes requirements

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
        |set 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
        |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;

case 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.":
  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
      |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.":
  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
        |set to Off.":
  assume regulator_mode == Isolette_Data_Model::Regulator_Mode.Failed_Regulator_Mode;
  guarantee heat_control == Isolette_Data_Model::On_Off.Off;
```





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

OSATE AADL Editor

HAMR SysMLv2 Front-end

NEW: We developed a VSCode SysMLv2 HAMR front-end based on the SysIDE VSCode plug-in (prototype)

SysMLv2 component interfaces

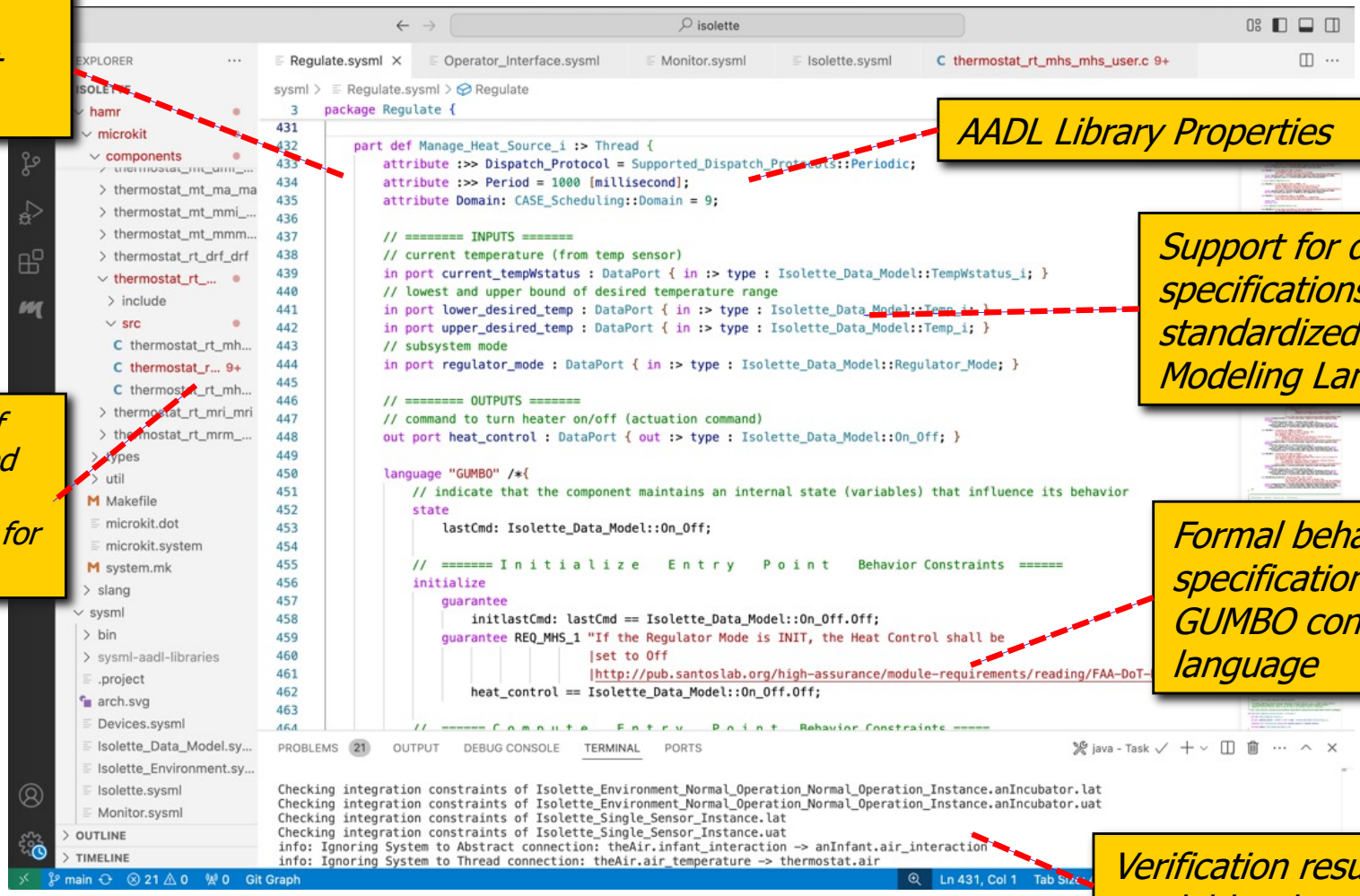
AADL Library Properties

Support for data type specifications in the standardized AADL Data Modeling Language

Integration of MicroKit-based Rust and C development for seL4

Formal behavior specifications in GUMBO contract language

Verification results for model-level contracts

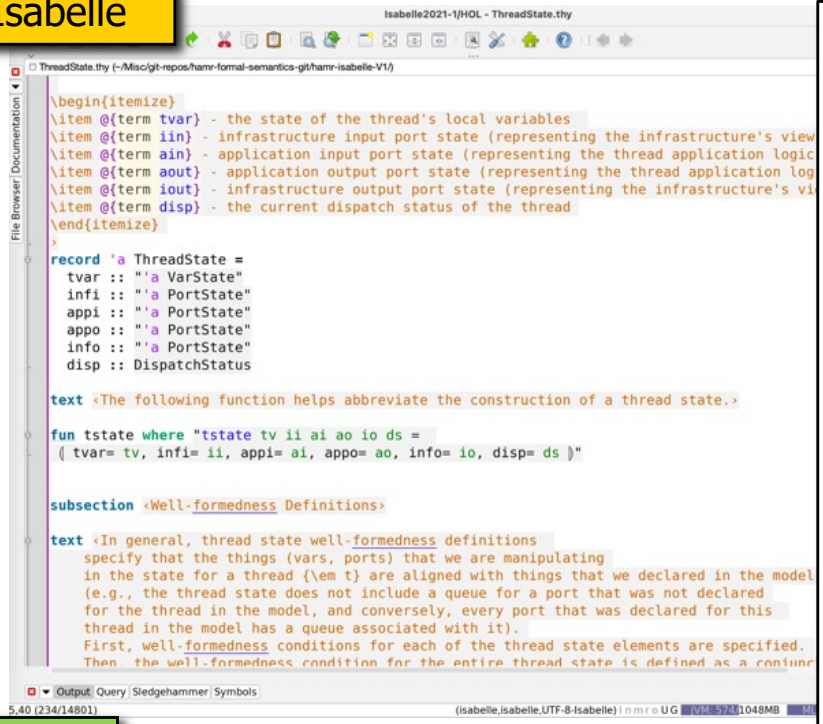


AADL / HAMR Formal Semantics

140+ page literate-style Isabelle/HOL theories for AADL/SysMLv2 HAMR execution model (guides our design of our contracts and verification/testing framework)

Joint work with Stefan Hallerstede (U. Aarhus)

Isabelle



Latex/PDF generated from Isabelle

record *'a ThreadState* =
tvar :: *'a VarState*
infi :: *'a PortState*
appi :: *'a PortState*
appo :: *'a PortState*
info :: *'a PortState*
disp :: *DispatchStatus*

The following function helps abbreviate the construction of a thread state.

fun *tstate* **where** *tstate tv ii ai ao io ds* =
(| *tvar*= *tv*, *infi*= *ii*, *appi*= *ai*, *appo*= *ao*, *info*= *io*, *disp*= *ds* |)

2.4.2 Well-formedness Definitions

In general, thread state well-formedness definitions specify that the things (vars, ports) that we are manipulating in the state for a thread *t* are aligned with things that we declared in the model for *t*. (e.g., the thread state does not include a queue for a port that was not declared for the thread in the model, and conversely, every port that was declared for this thread in the model has a queue associated with it). First, well-formedness conditions for each of the thread state elements are specified. Then, the well-formedness condition for the entire thread state is defined as a conjunction of these properties.

Well-formed Thread State Elements

definition *wf-ThreadState-tvar*:: *Model* ⇒ *CompId* ⇒ (*'a VarState*) ⇒ *bool* **where**
wf-ThreadState-tvar m c vs ≡ *wf-VarState vs* {*v* . *isVarOfCID m c v*}

The infi component of a ThreadState (input infrastructure port map) is well formed when the domain of the infi port map is equal to the set of input ports for the thread declared in the model. Intuitively, each of the declared "in" ports for the thread (according to the model) is associated with a infrastructure message queue, (and there are no "extra" ports in the map).

definition *wf-ThreadState-infi*:: *Model* ⇒ *CompId* ⇒ (*'a PortState*) ⇒ *bool* **where**
wf-ThreadState-infi m c ps ≡ *wf-PortState ps* {*p* . *isInCIDPID m c p*}

The definitions below for other port-state elements are similar.

definition *wf-ThreadState-appi*:: *Model* ⇒ *CompId* ⇒ (*'a PortState*) ⇒ *bool* **where**
wf-ThreadState-appi m c ps ≡ *wf-PortState ps* {*p* . *isInCIDPID m c p*}

definition *wf-ThreadState-appo*:: *Model* ⇒ *CompId* ⇒ (*'a PortState*) ⇒ *bool* **where**
wf-ThreadState-appo m c ps ≡ *wf-PortState ps* {*p* . *isOutCIDPID m c p*}

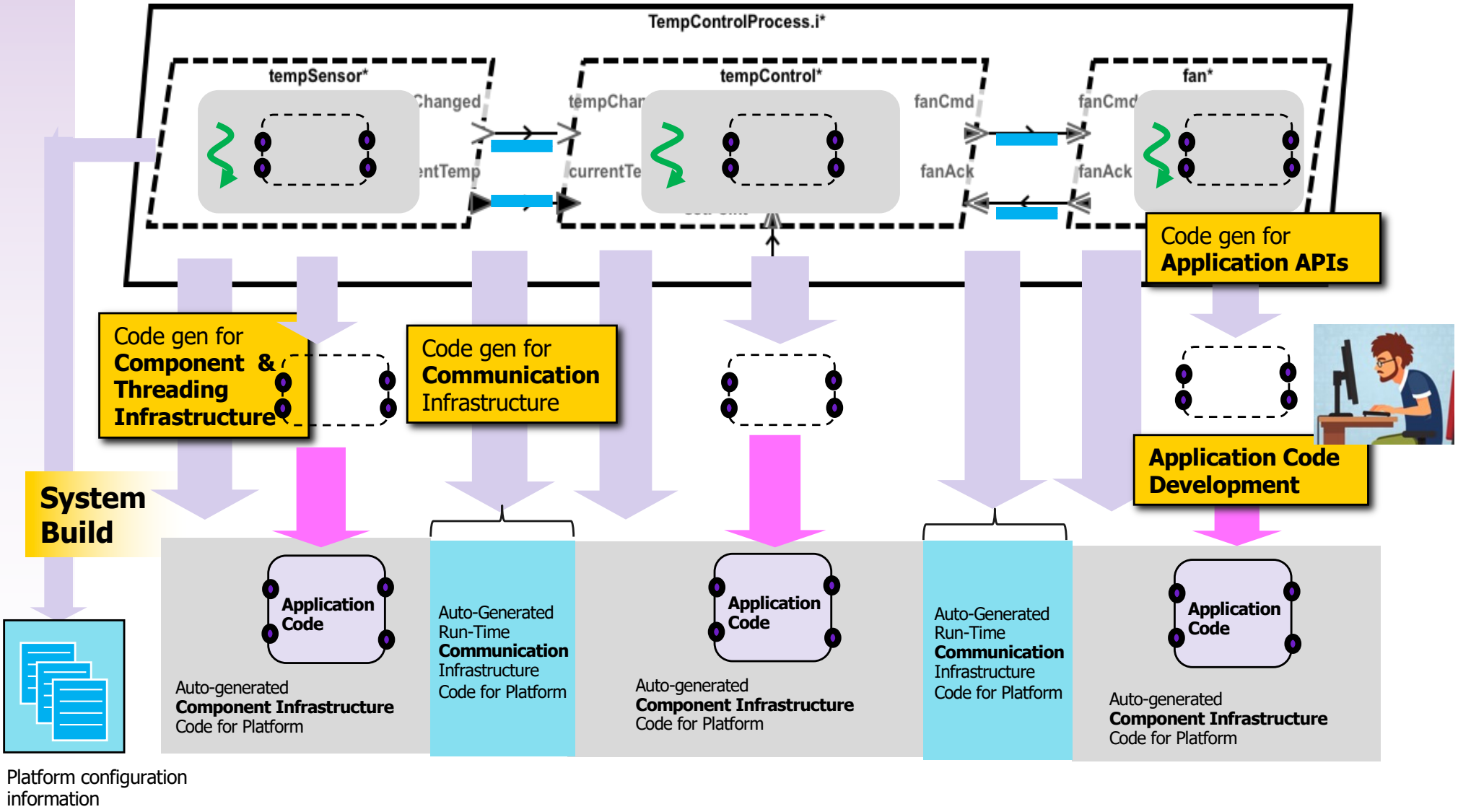
definition *wf-ThreadState-info*:: *Model* ⇒ *CompId* ⇒ (*'a PortState*) ⇒ *bool* **where**
wf-ThreadState-info m c ps ≡ *wf-PortState ps* {*p* . *isOutCIDPID m c p*}

PROVERS

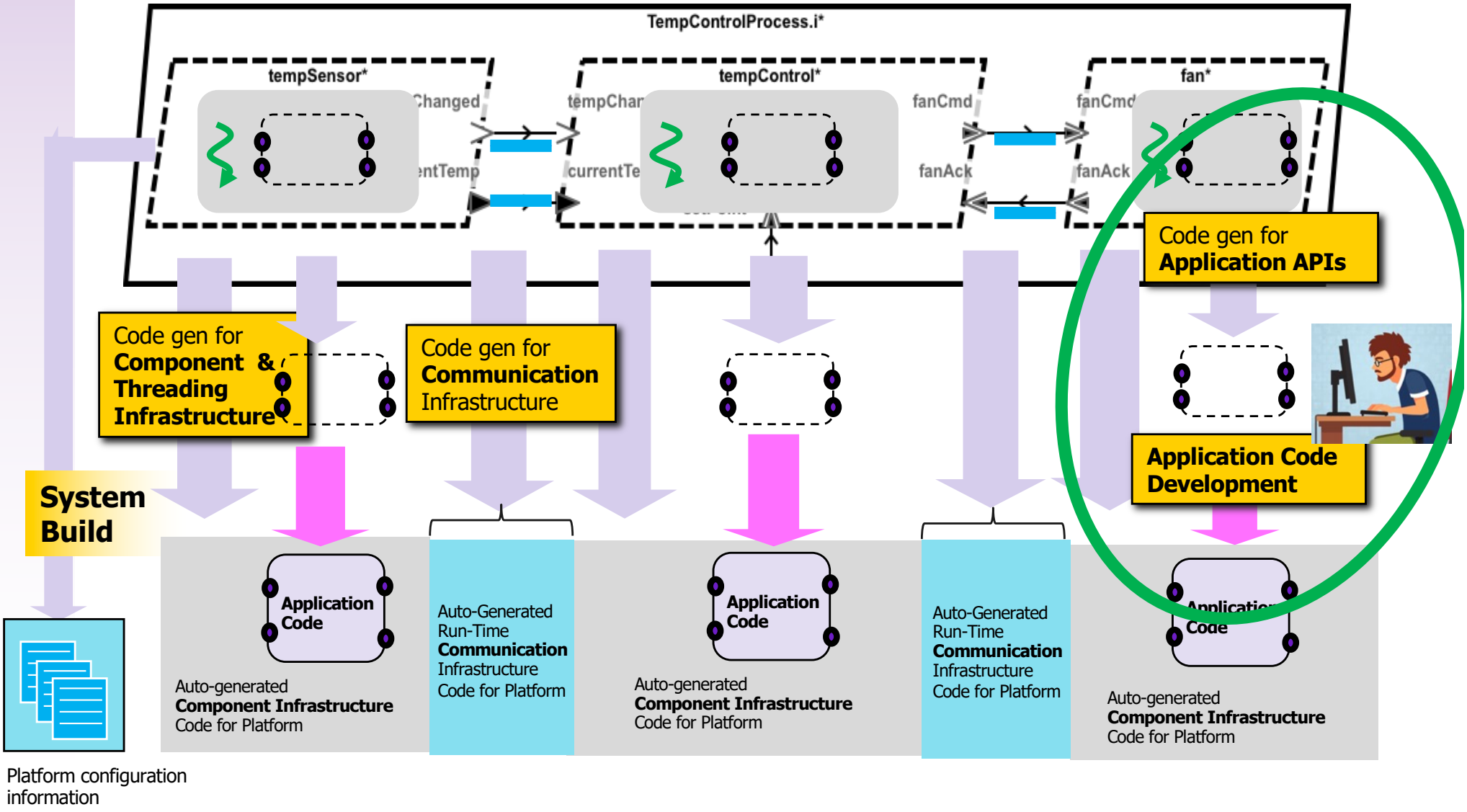
- Enhanced and scope expanded
- Prove soundness of contract framework
- Extend formalization downwards towards seL4 proof-base

Note limited scope: HAMR subset of AADL/SysMLv2; run-time semantics; connection to code generator by manual inspection

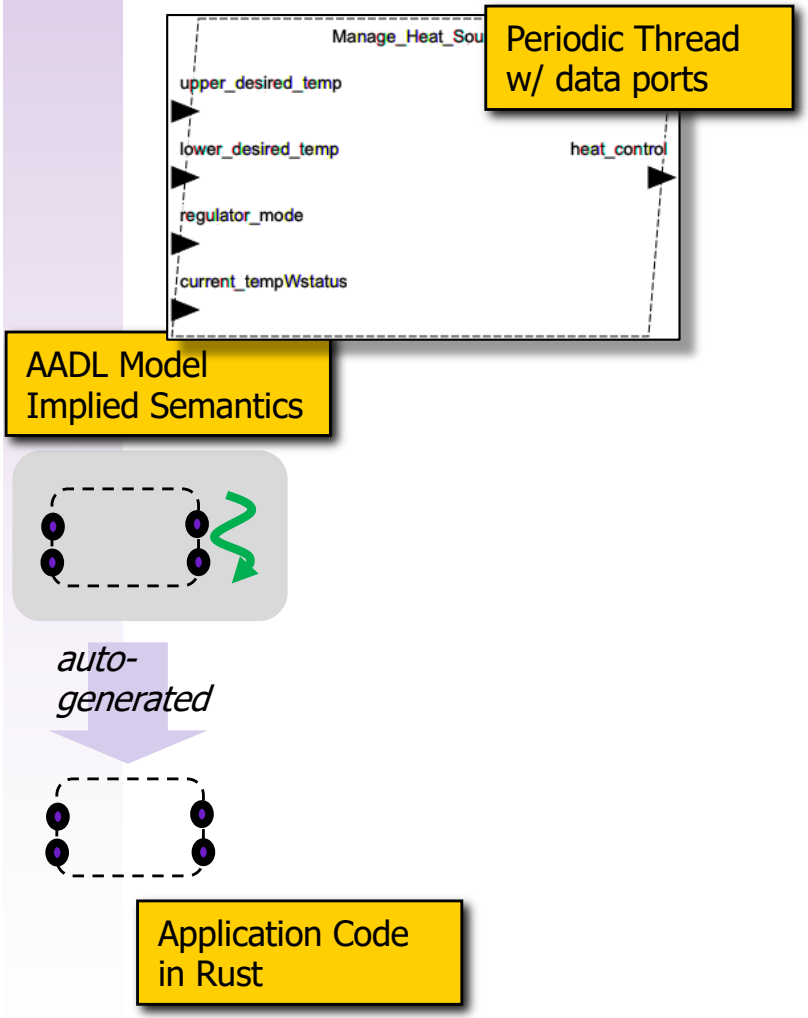
HAMR Code Generation



HAMR Code Generation

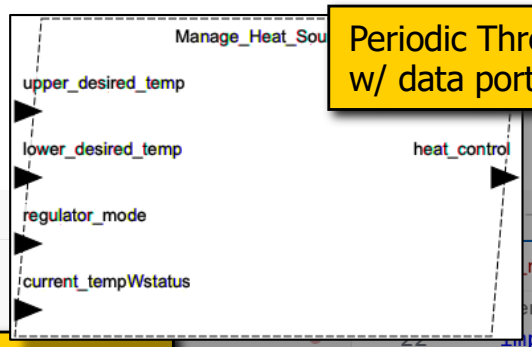


Component Application Code Interfaces Generated from SysMLv2/AADL Model



*……Interfaces/APIs/Skeletons **+ contracts** for application code are auto-generated from SysMLv2/AADL model.*

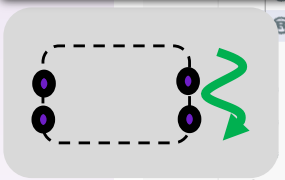
Component Application Code Interfaces Generated from SysMLv2/AADL Model



Periodic Thread w/ data ports

*……Interfaces/APIs/Skeletons **+ contracts** for application code are auto-generated from SysMLv2/AADL model.*

AADL Model Implied Semantics



auto-generated

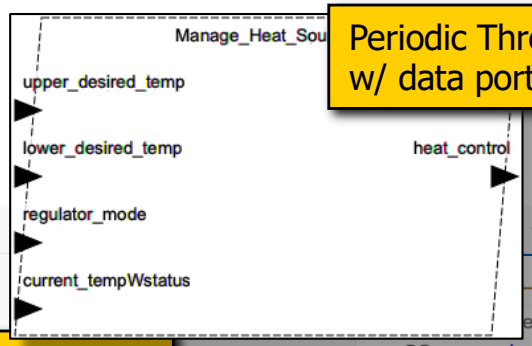
Application Code in Rust

Component contract (small excerpt)

Skeleton for application code entry point

Verus error indicates that contract is not yet satisfied

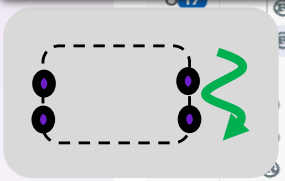
Component Application Code Interfaces Generated from SysMLv2/AADL Model



Periodic Thread w/ data ports

*...Developer **adds application code** to contract-annotated skeleton, and **verification/testing tools check conformance to contracts**.*

AADL Model Implied Semantics



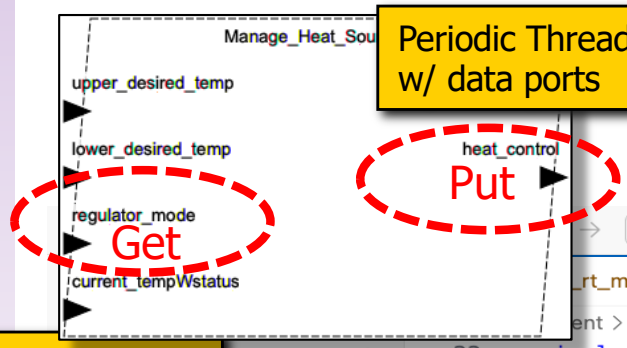
auto-generated

Application Code in Rust

Adding application code to skeleton

```
22  impl thermostat_rt_mhs_mhs {
56      pub fn timeTriggered<API: thermostat_rt_mhs_mhs_Full_Api>(
103         (old(api).regulator_mode == Isolette_Data_Model::Regulator_Mode::Failed_Regulat
104           (api.heat_control == Isolette_Data_Model::On_Off::Off)
105         // END_MARKER_TIME_TRIGGERED_ENSURES
106     {
107       // -------------- Get values of input ports ------------------
108       let lower: Temp_i = api.get_lower_desired_temp();
109       let upper: Temp_i = api.get_upper_desired_temp();
110       let regulator_mode: Regulator_Mode = api.get_regulator_mode();
111       let currentTemp: TempWstatus_i = api.get_current_tempWstatus();
112
113       //================ compute / control logic ============================
114
115       // current command defaults to value of last command (REQ-MHS-4)
116       let mut currentCmd: On_Off = self.lastCmd;
117
118       match regulator_mode {…
146
147       // -------------- Set values of output ports ------------------
148       api.put_heat_control(currentCmd);
149       self.lastCmd = currentCmd
150   } fn timeTriggered
151
```

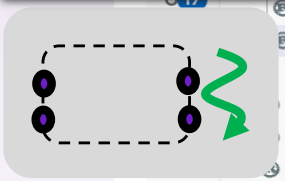
Component Application Code Interfaces Generated from SysMLv2/AADL Model



Periodic Thread w/ data ports

*...Developer uses auto-generate APIs to **get** and **put** data on component ports*

AADL Model Implied Semantics



auto-generated

Application Code in Rust

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

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

Verus indicates that contract is satisfied

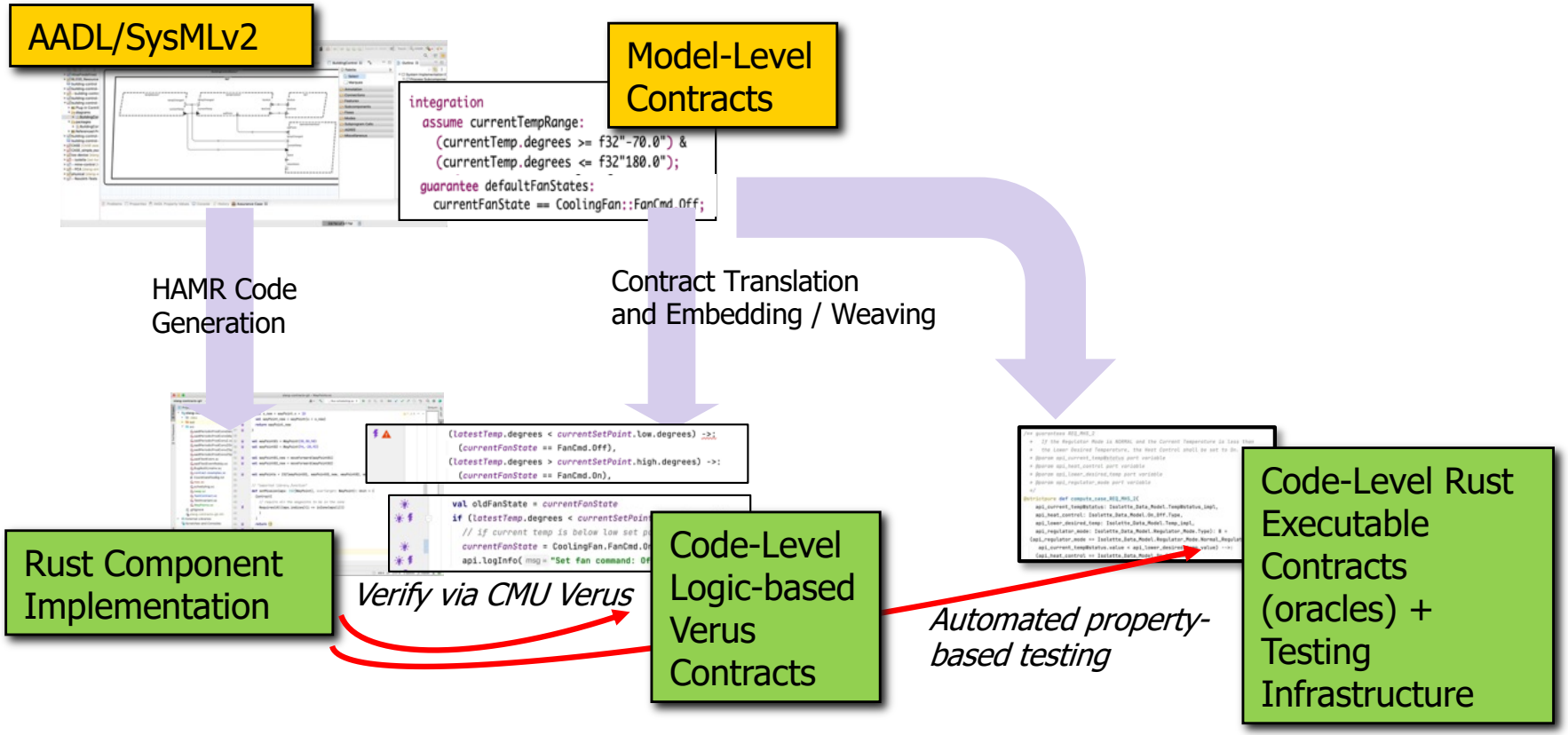
```
impl thermostat_rt_mhs_mhs {
    pub fn timeTriggered<API: thermostat_rt_mhs_mhs_Full_Api>(
        (old(api).regulator_mode == Isolette_Data_Model::Regulator_Mode::Failed_Regulat
          (api.heat_control == Isolette_Data_Model::On_Off::Off)
        // END MARKER TIME TRIGGERED ENSURES
    {
      // -------------- Get values of input ports ----------------
      let lower: Temp_i = api.get_lower_desired_temp();
      let upper: Temp_i = api.get_upper_desired_temp();
      let regulator_mode: Regulator_Mode = api.get_regulator_mode();
      let currentTemp: TempWstatus_i = api.get_current_tempWstatus();

      //================ compute / control logic ============================

      // current command defaults to value of last command (REQ-MHS-4)
      let mut currentCmd: On_Off = self.lastCmd;

      match regulator_mode {…

      // -------------- Set values of output ports ------
      api.put_heat_control(currentCmd);
      self.lastCmd = currentCmd
    } fn timeTriggered
```

PROBLEMS OUTPUT DEBUG CONSOLE

No problems have been detected in the workspace.

Integrated Model/Code Contracts

Extend existing Slang-based framework to support Rust..



Automatically Embedded Rust/Verus Logical Contracts

Verification of Rust application code against contracts using Verus (excerpts)

Pre

Post

```
57      pub fn timeTriggered<API: thermostat_rt_mhs_mhs_Full_Api>(
58        &mut self,
59        api: &mut thermostat_rt_mhs_mhs_Application_Api<API>)
          requires
            // BEGIN MARKER TIME TRIGGERED REQUIRES
62          // assume lower_is_lower_temp
63          old(api).lower_desired_temp.degrees <= old(api).upper_desired_temp.degrees
            // END MARKER TIME TRIGGERED REQUIRES
          ensures
66          // BEGIN MARKER TIME TRIGGERED ENSURES
67          // guarantee lastCmd
```

```
// 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.
((old(api).regulator_mode == Regulator_Mode::Normal_Regulator_Mode) &&
  (old(api).current_tempWstatus.degrees < old(api).lower_desired_temp.degrees)) ==>
 (api.heat_control == On_Off::Onn),
```

```
76          //   If the Regulator Mode is NORMAL and the Current Temperature is less than
77          //   the Lower Desired Temperature, the Heat Control shall be set to On.
78          ((old(api).regulator_mode == Regulator_Mode::Normal_Regulator_Mode) &&
79            (old(api).current_tempWstatus.degrees < old(api).lower_desired_temp.degrees)) ==>
80          (api.heat_control == On_Off::Onn),
```

... (clauses for Reqs 3-5 omitted)

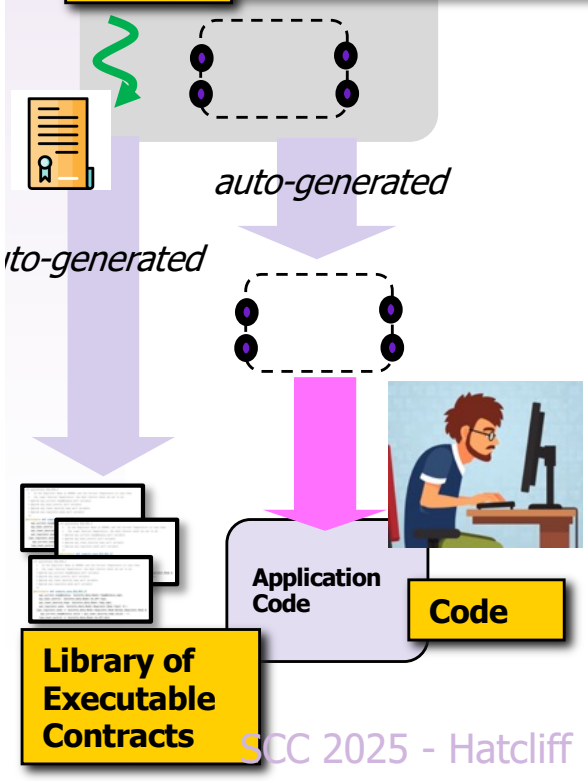
Manage Heat Source Contracts

Each clause in **model-level** GUMBO contracts is translated to a **code-level** Boolean function in Rust that works on the appropriate port/thread state elements

AADL GUMBO Contract (clause REQ_MHS_2)

```
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



Rust Executable Contract (clause REQ_MHS_2)

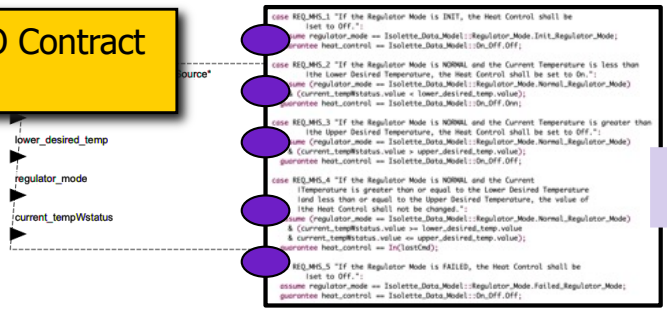
```
/** guarantee REQ_MHS_2
  *   If the Regulator Mode is NORMAL and the Current Temperature is less than
  *   the Lower Desired Temperature, the Heat Control shall be set to On.
  * @param api_current_tempWstatus incoming data port
  * @param api_lower_desired_temp incoming data port
  * @param api_regulator_mode incoming data port
  * @param api_heat_control outgoing data port
  */
pub fn compute_case_REQ_MHS_2(
  api_current_tempWstatus: TempWstatus_i,
  api_lower_desired_temp: Temp_i,
  api_regulator_mode: Regulator_Mode,
  api_heat_control: On_Off) -> bool
{
  implies(
    lhs: lhs: (api_regulator_mode == Regulator_Mode::Normal_Regulator_Mode) &
      (api_current_tempWstatus.degrees < api_lower_desired_temp.degrees),
    rhs: rhs: api_heat_control == On_Off::Onn)
}
```

Traceability info automatically embedded

Manage Heat Source Contracts

Complete **Model-level** GUMBO contracts are translated to a hierarchy of executable Boolean functions in Rust (**code-level)**

Model: AADL GUMBO Contract (all five clauses)



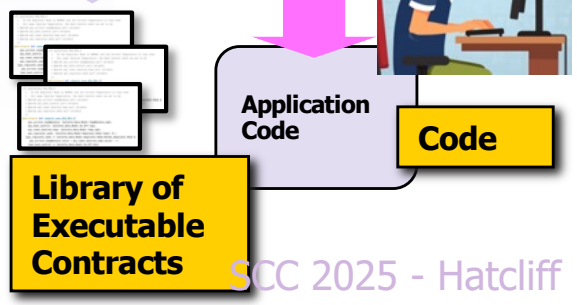
*Code generation **weaves** together functions for contract clauses **to form a pre-condition checker and a post-condition checker** (also includes data invariants, etc.).*

(see paper for details)

Model

auto-generated

ıto-generated



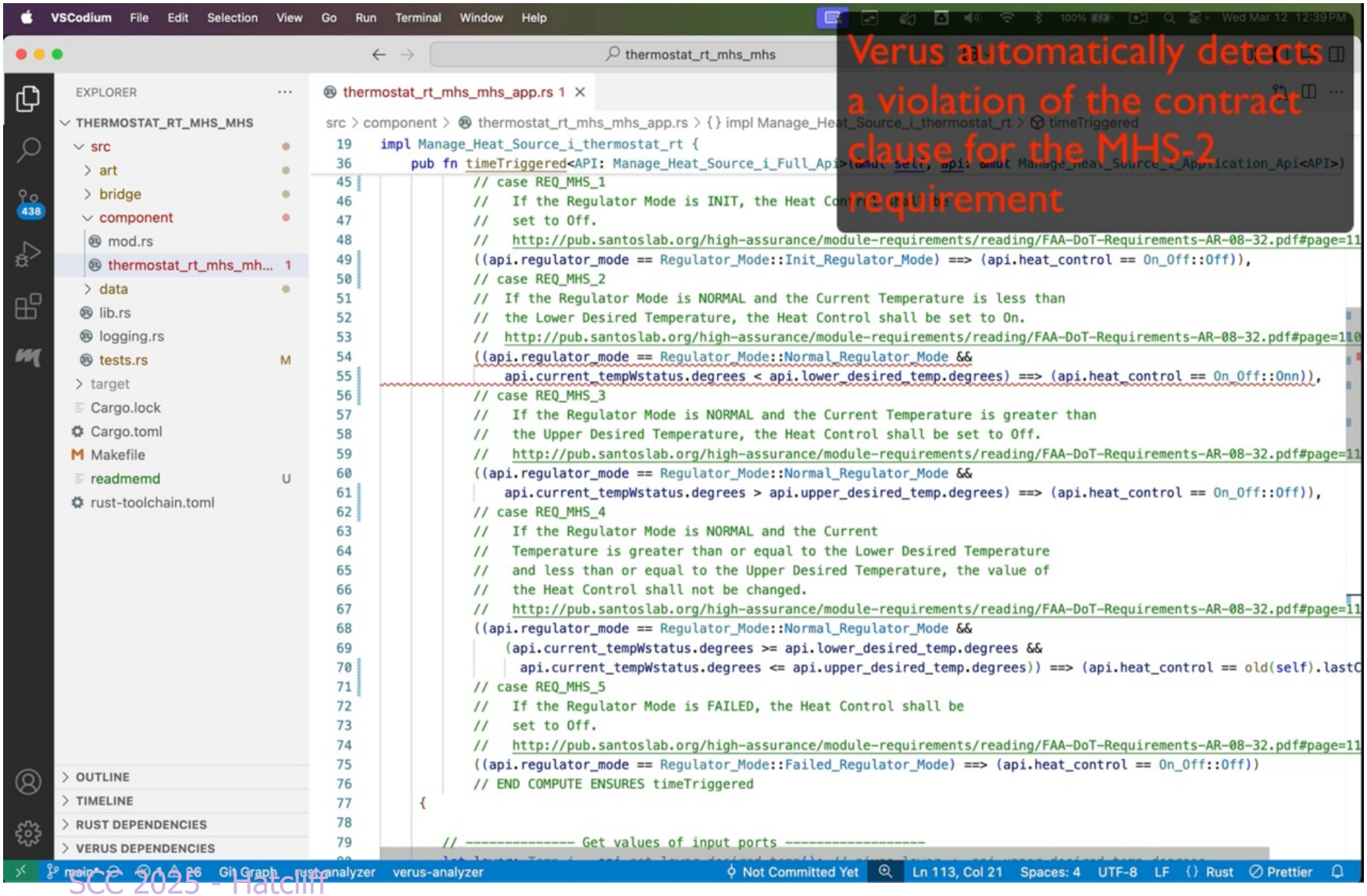
Code: Auto-generated in Testing Library

Rust Executable Contract (aggregated clauses for post-condition)

```
  In_lastCmd: On_Off,
  api_current_tempWstatus: TempWstatus_i,
  api_lower_desired_temp: Temp_i,
  api_regulator_mode: Regulator_Mode,
  api_upper_desired_temp: Temp_i,
  api_heat_control: On_Off) -> bool
{
  let r0: bool = compute_case_REQ_MHS_1(api_regulator_mode, api_heat_control);
  let r1: bool = compute_case_REQ_MHS_2(api_current_tempWstatus, api_lower_des:
  let r2: bool = compute_case_REQ_MHS_3(api_current_tempWstatus, api_regulator_
  let r3: bool = compute_case_REQ_MHS_4(In_lastCmd, api_current_tempWstatus, ap
  let r4: bool = compute_case_REQ_MHS_5(api_regulator_mode, api_heat_control);

  return r0 && r1 && r2 && r3 && r4;
}
```

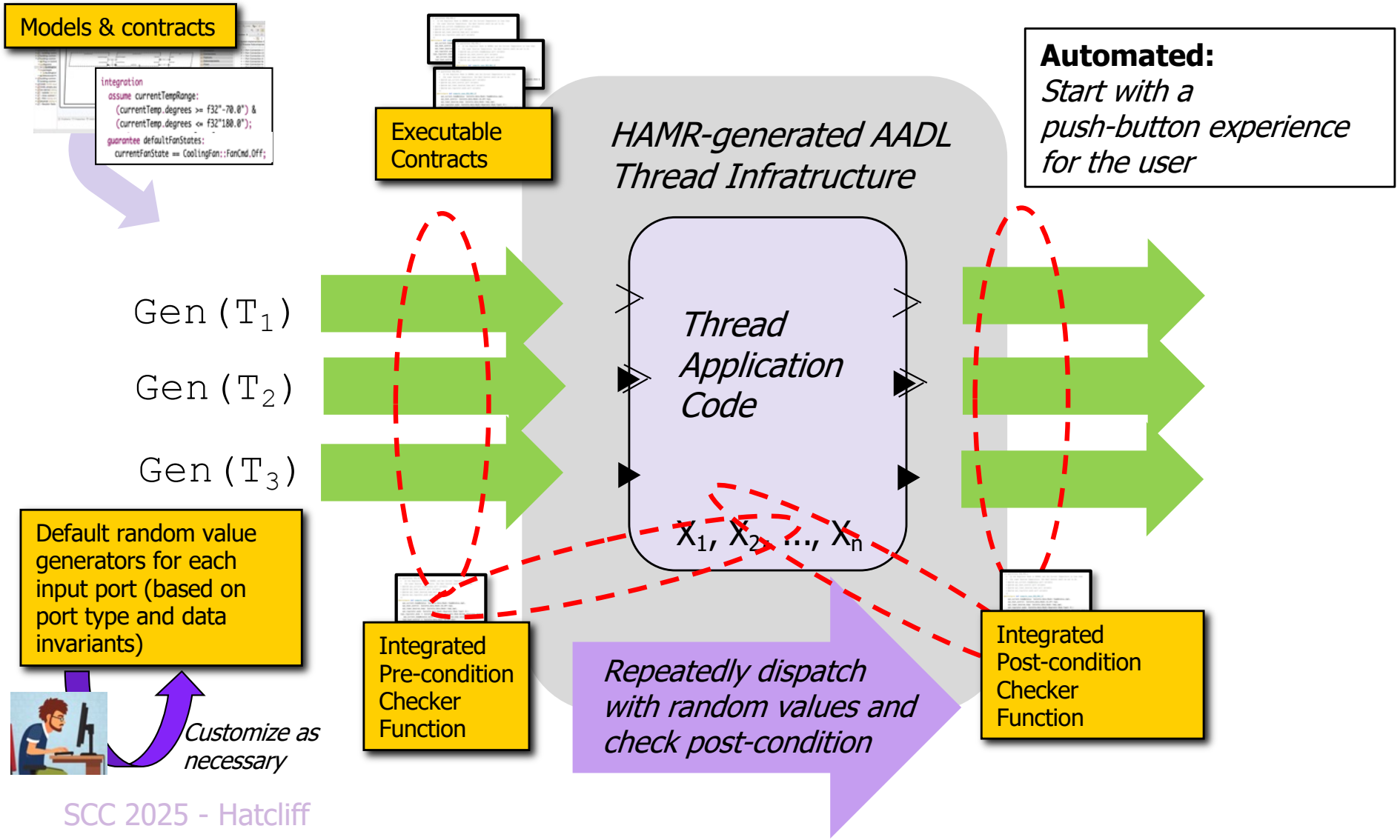
Demo

Verification against contracts using Verus verification tool…



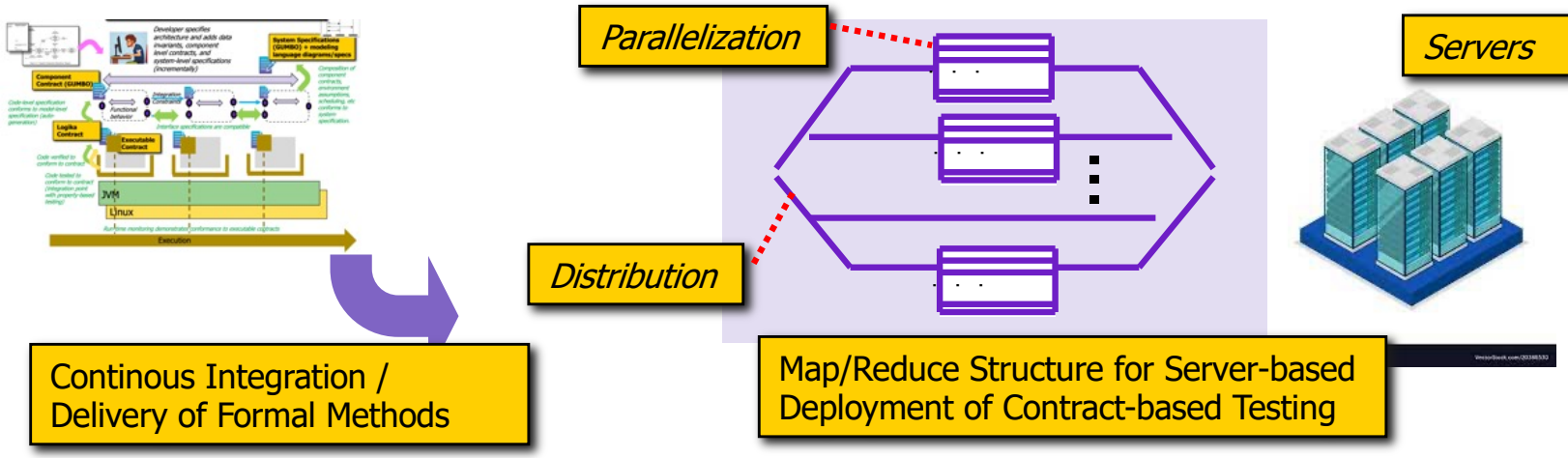
Auto-Generated Property-based Testing Harness

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.



Scaling Up - Property-based Testing Server-Based Deployment

For Slang property-based testing, HAMR generates a server-based deployment to run the framework in a distributed/parallel fashion...

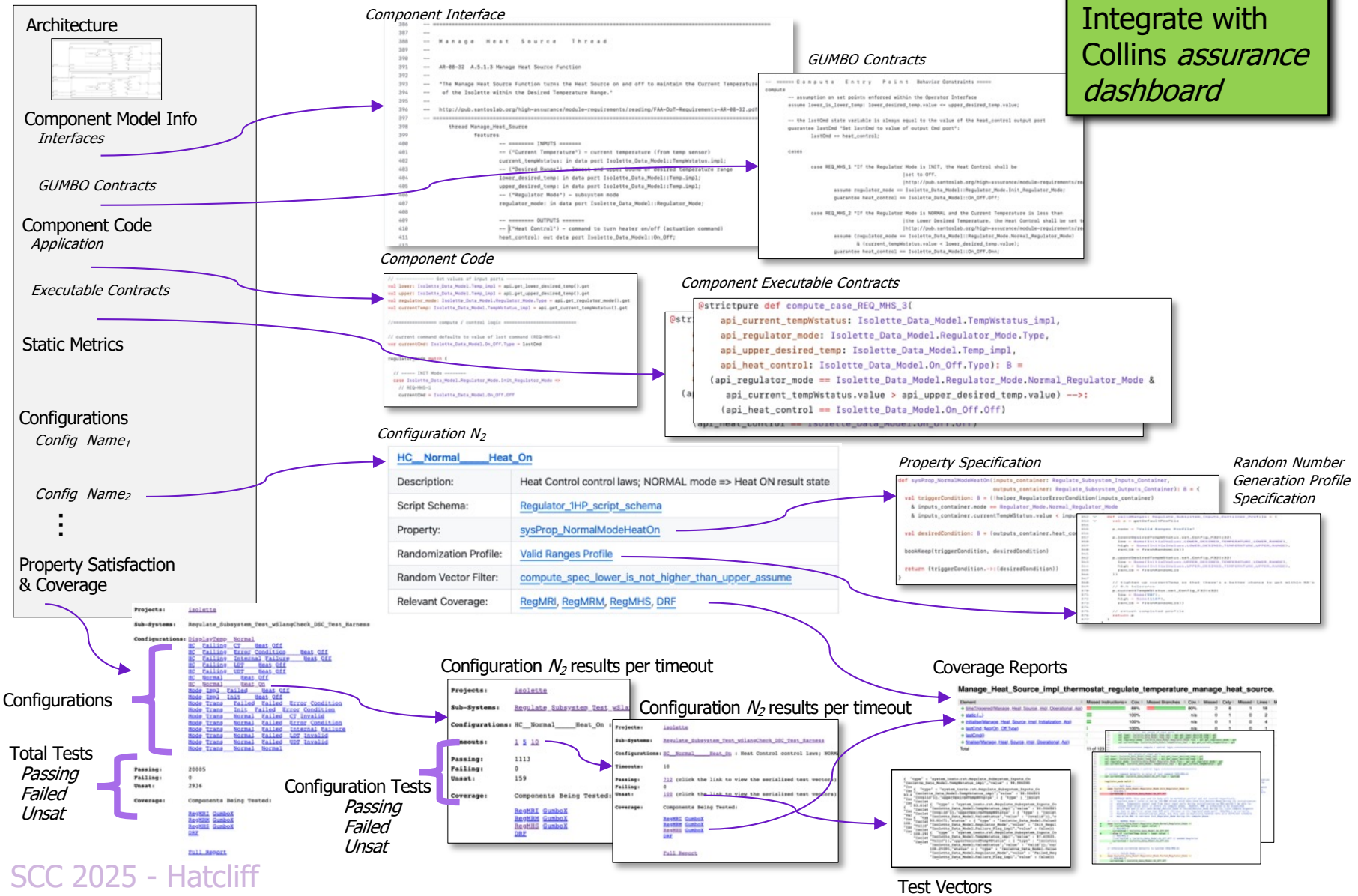


- Random generators and contract-based tests are farmed out to a configurable family of servers
- Test vectors and results are serialized for flexible deployment, reporting, and replay of the tests
- Currently hosted using our Jenkins setup, but easy for HAMR to automatically generate deployment scripts,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

Extensive Assurance Artifacts

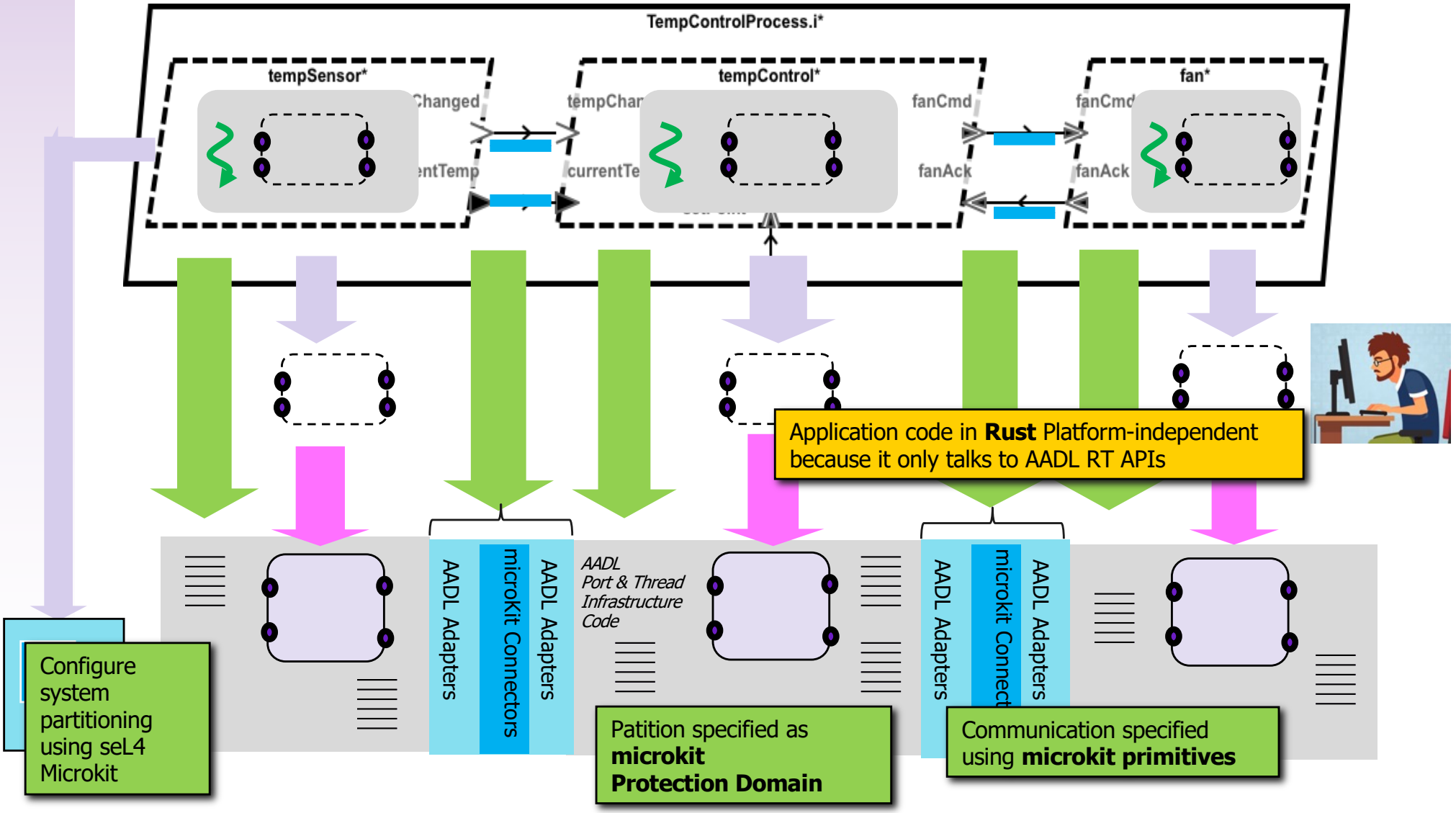
HAMR provides extensive auto-generation and reporting of assurance artifacts

PROVERS:
Integrate with Collins *assurance dashboard*



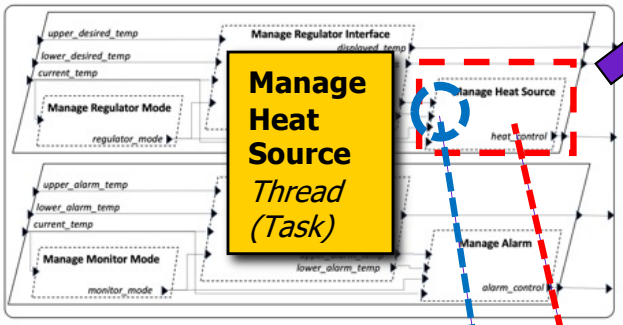
HAMR Code Generation seL4 Platform

HAMR instantiation for **Rust** - based development on **SeL4 microkernel** using seL4 microKit



Microkit Kernel Configuration (excepts)

HAMR auto-generated platform artifacts for seL4 include microKit *system description file (XML)* that specifies the configuration of kernel partitions, allowed communication pathways, resource allocation, etc.



Manage Heat Source *Thread (Task)*

HAMR generation

Partition (protection domain) for component

Memory + permissions for port queue

`microkit.system` *(excerpts)*

```
...
<protection_domain name="thermostat_rt_mhs_mhs_MON" domain="domain_9">
  <program_image path="thermostat_rt_mhs_mhs_MON.elf" />
  <protection_domain name="thermostat_rt_mhs_mhs" domain="domain_9" id="1">
    <program_image path="thermostat_rt_mhs_mhs.elf" />
    <map mr="Isolette_Single_Sensor_Instance_thermostat_rt_mri_mri_upper_desired_temp_1_Memory_Region"
         vaddr="0x10_000_000"
         perms="r"
         setvar_vaddr="upper_desired_temp_queue_1"
    />
    <map mr="Isolette_Single_Sensor_Instance_thermostat_rt_mri_mri_lower_desired_temp_1_Memory_Region"
         vaddr="0x10_001_000"
         perms="r"
         setvar_vaddr="lower_desired_temp_queue_1"
    />
    <map mr="Isolette_Single_Sensor_Instance_thermostat_rt_mhs_mhs_heat_control_1_Memory_Region"
         vaddr="0x10_002_000"
         perms="rw"
         setvar_vaddr="heat_control_queue_1"
    />
    <map mr="Isolette_Single_Sensor_Instance_thermostat_rt_mrm_mrm_regulator_mode_1_Memory_Region"
         vaddr="0x10_003_000"
         perms="r"
         setvar_vaddr="regulator_mode_queue_1"
    />
    <map mr="Isolette_Single_Sensor_Instance_temperature_sensor_cpi_thermostat_current_tempWstatus_1_Memory_Region"
         vaddr="0x10_004_000"
         perms="r"
         setvar_vaddr="current_tempWstatus_queue_1"
    />
  </protection_domain>
</protection_domain>
...
```

Conclusion

- Protect critical systems from malicious attacks using **verified micro-kernels**
- **Automate** and **control** the development process using a model-based development approach
- Provide a **multi-level contract framework** that supports seamless transition between automated testing and verification
- **Models as sound/faithful abstractions** of deployed systems support understanding, simplify development, provide basis for many forms of analysis
- **Semantic foundation** (e.g., Isabelle formalization) and **abstraction layers** in run-time architecture enables new backends to be added while achieving consistent semantics
- **Integrated assurance case generation** framework

Resources

Sireum HAMR

High Assurance Model-based Rapid Engineering of Embedded Systems

Publicly available tool:
http://hamr.sireum.org

Resources on HAMR web site

- Distribution available for Windows, Linux, and Mac (also virtualized) hamr.sireum.org
- Documentation, examples, and tutorial material for HAMR
- Educational resources -- slides, recorded lectures, and guided exercises for HAMR Slang back end

SAnToS Artifacts -- Formal Methods in Action

Isolette

-- AN INFANT INCUBATOR

- Online resources for Isolette artifacts