Model-based Development for seL4 Microkit/Rust with Integrated Formal Methods using HAMR

seL4 Summit – Sept 3, 2025

Kansas State University

John Hatcliff
Robby
Jason Belt

Aarhus University

Stefan Hallerstede

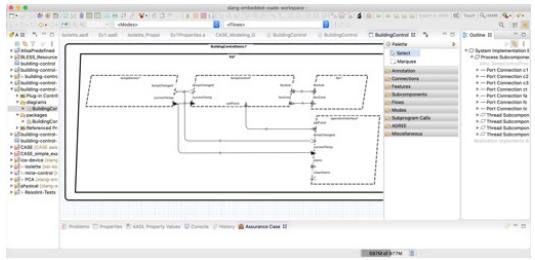
With collaborators at …

Collins Aerospace
Dornerworks
UNSW
Proofcraft
Carnegie Mellow Univ.
Univ. of Kansas

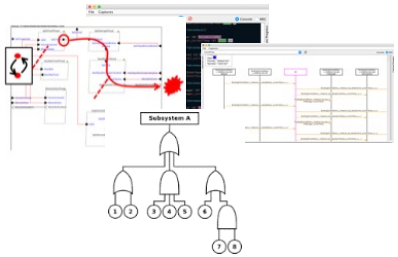
HAMR

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

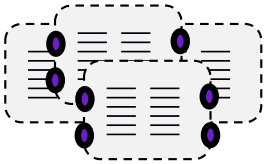
Modeling, analysis, and verification in the **SysMLv2** and **AADL** modeling languages



Model-level behavior specifications (e.g., contracts) and analyses



Component development, automated testing, and verification in multiple languages



- C
- Rust with Verus verification
- Slang (developed at Kansas State)
  - safety-critical subset of Scala
  - contract-based verification
  - transpiles to C and Rust

Deployments aligned with AADL run-time on multiple platforms

JVM Deployment

JVM (Java Virtual Machine)

Linux Deployment

Linux

seL4 Deployment

seL4 Security. Performance. Proof.

verified micro-kernel

CAmkES & microKit

Potential Benefits to seL4 Application Developers

A systems engineering environment based on standardized modeling languages (SysMLv2, AADL) with accompanying analysis, verification, and assurance case tools

Industrial workflows

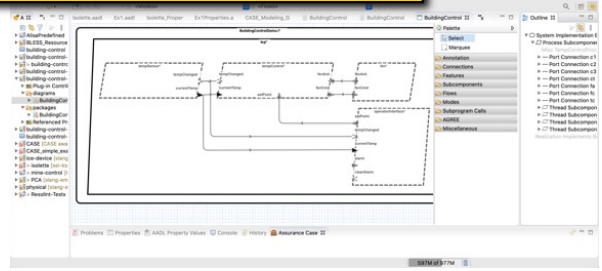
Requirements Engineering

Planning Diagrams

Domain Concepts

Systems Engineering

SysMLv2 / AADL IDE – VSCode / Cameo

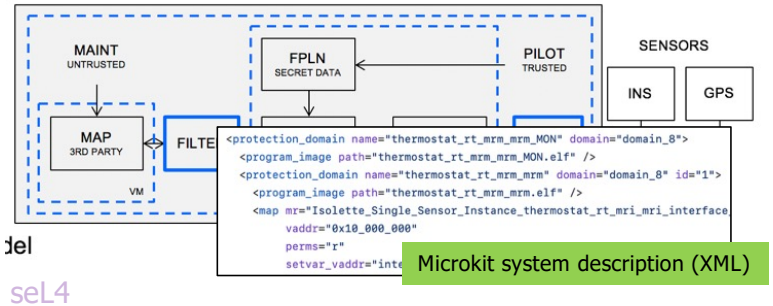


..integrate seL4 configuration with…

Software/Hardware/Middleware modeling
Architecture constraints / patterns
Component contracts + verification
Assurance case construction
..
Information Flow Spec & Analysis
Timing / Scheduleability

seL4 Deployment

Microkit / CAmKES Specification



Microkit system description (XML)

Existing seL4 ecosystem tools for component-oriented specification of seL4 capabilities, partitioning, and inter-partition communication

Potential Benefits to seL4 Application Developers

A systems engineering environment based on standardized modeling languages (SysMLv2, AADL) with accompanying analysis, verification, and assurance case tools

Industry workflows

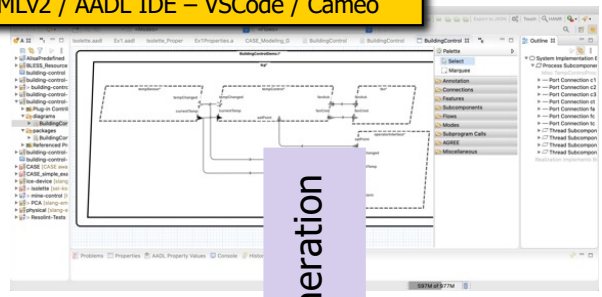
Requirements Engineering

Planning Diagrams

Domain Concepts

Systems Engineering

SysMLv2 / AADL IDE – VSCode / Cameo



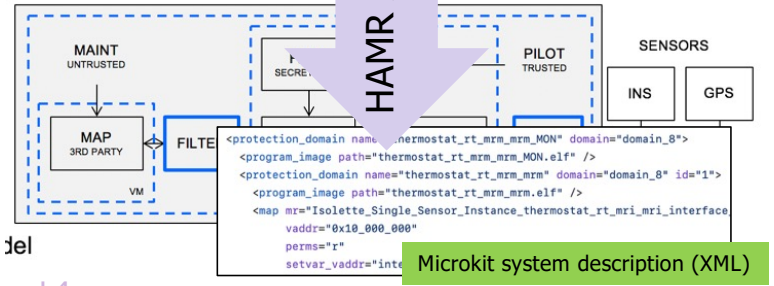
HAMR Auto-generation

INSPECTA Capabilities

Software/Hardware/Middleware modeling
Architecture constraints / patterns
Component contracts + verification
Assurance case construction
..
Information Flow Analysis
Timing / Scheduleability

Microkit / CAmk pecification

seL4 Deployment



Microkit system description (XML)

Auto-generated

Microkit system description
Microkit code bindings (entry point skeletons)
VM configurations (partial)
Build scripts
Skeleton system,
e.g., immediately runnable on Qemu (agile)

Protection domain automated unit testing infrastructure
Code-level contract information (e.g., Rust/Verus)

Traceability information
Assurance case structures (partial)

Goals:
** Make seL4 more accessible*
** Framework for engineering* ***verified applications*** *on top of seL4 verified infrastructure*

Context and Target Applications

On the DARPA PROVERS program, HAMR is being used to develop an experimental version of the mission computer for the Collins "**Launched Effects**" platform (final development will emphasize HAMR SysMLv2 to Rust)

Collins Aerospace

Launched Effects product line
= tube-launched, expendable UAVs

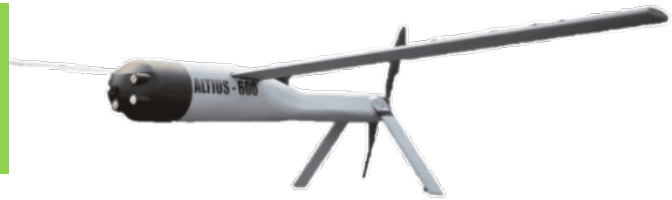
Launched Effects **Mission Computer**

..increase security and **modularity**

..decrease costs for development and assurance

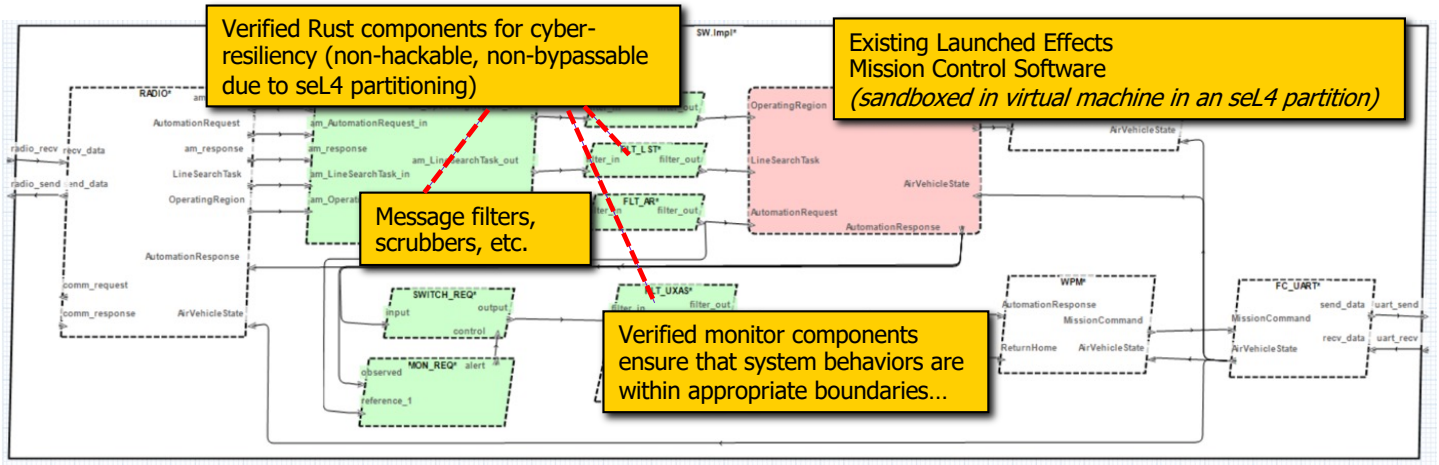
Video: https://youtu.be/SwPJHmZQMaM?si=NwTdb3VFpV-MxSre

DARPA PROVERS
= "integrating pipelines of formal methods in defense industry development processes"



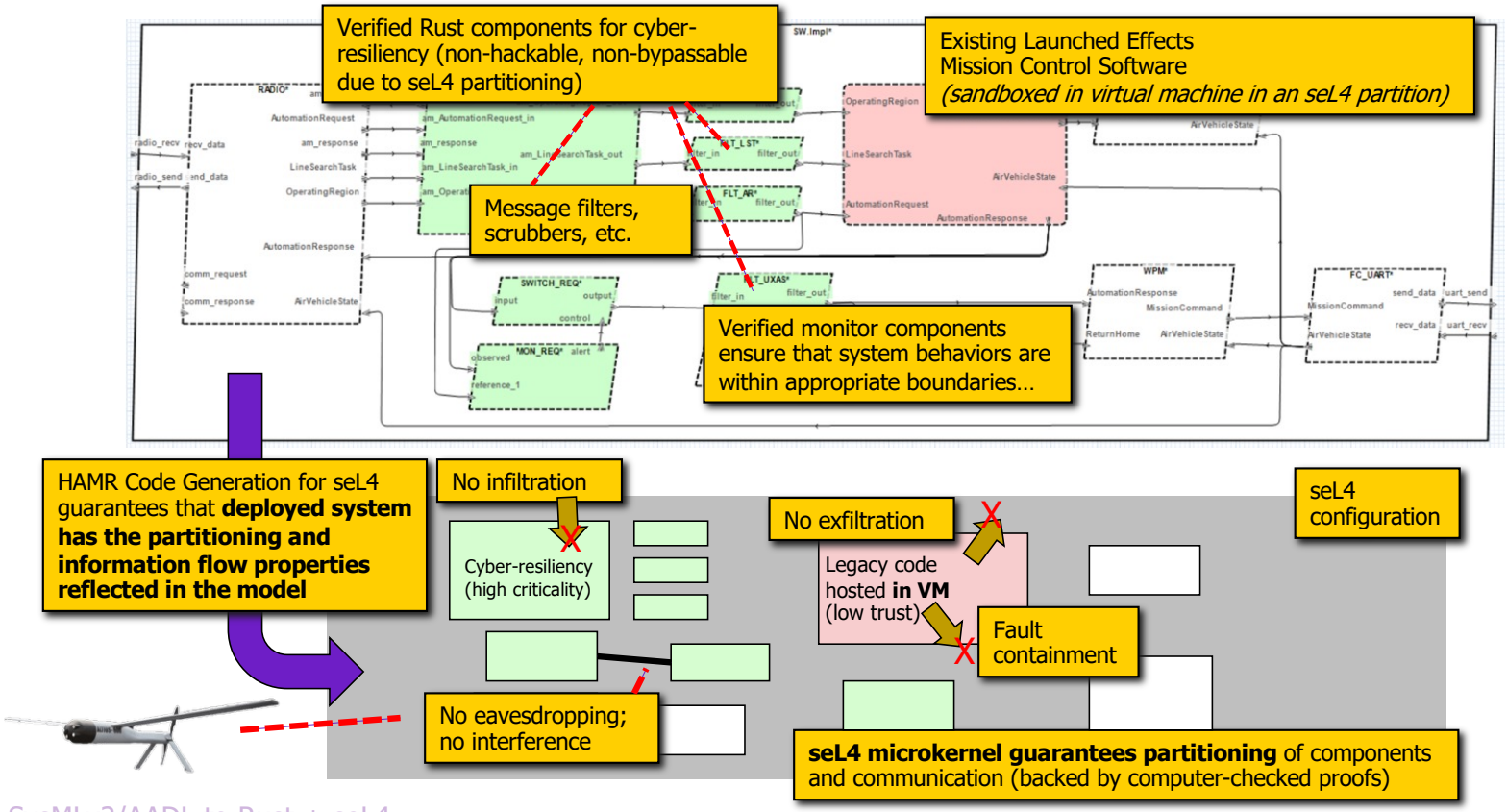
Characteristics of Supported Systems

Use HAMR SysMLv2/AADL modeling to specify partitioning, communication architecture of improved system



Characteristics of Supported Systems

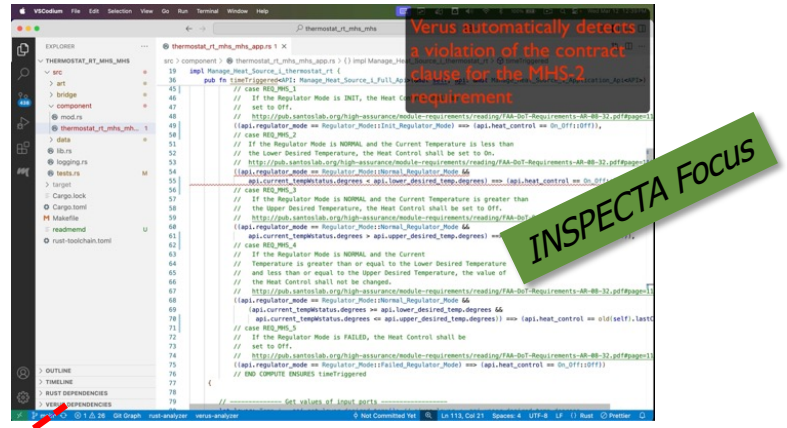
Use HAMR SysMLv2/AADL modeling to specify partitioning, communication architecture of improved system

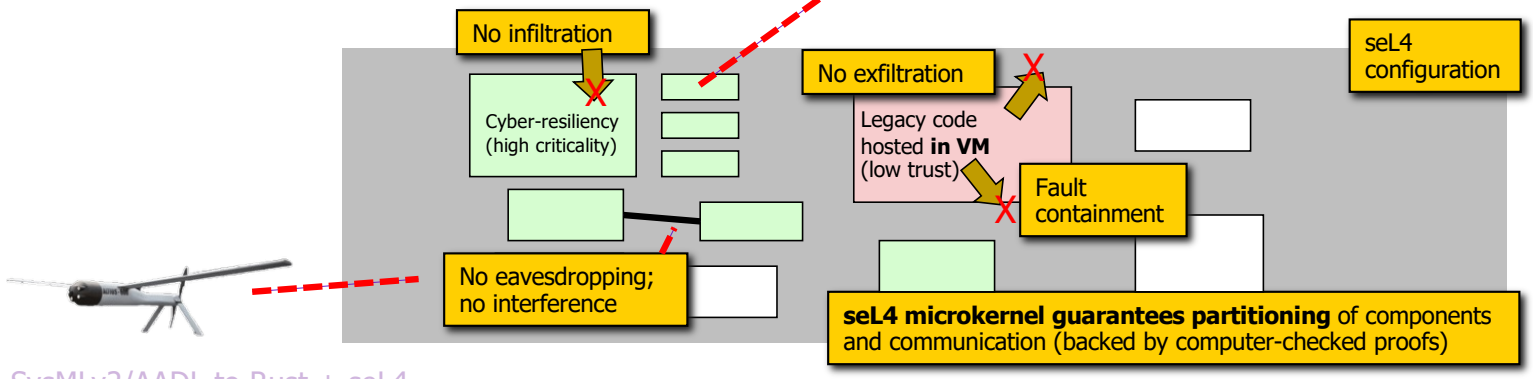


Verified Component Code

Rust (with Verus verification)

PROVERS program emphasis on memory safe languages…





Other Summit INSPECTA-Related Talks

Darren Cofer (Principal Investigator) –
Application to Collins Launched Effects

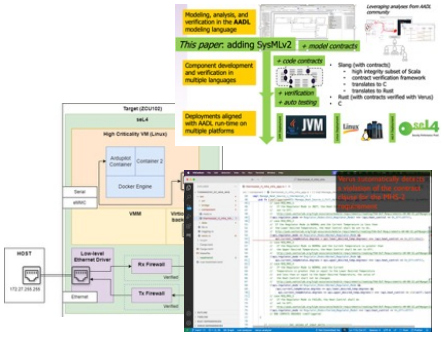
"Integration of seL4 in a Flight Vehicle Mission System"

Gerwin Klein –
Automating seL4 kernel correctness proofs for new platforms

"The next 700 verified seL4 platforms"

Gernot Heiser –
Verified infrastructure and services

"Trustworthy Systems R&D Update"



Robert VanVossen –
Rust contract-based development, testing, and verification of firewall components

"Rust-based drivers and verified rust applications on seL4"

Junming Zhao–
Verified infrastructure and services

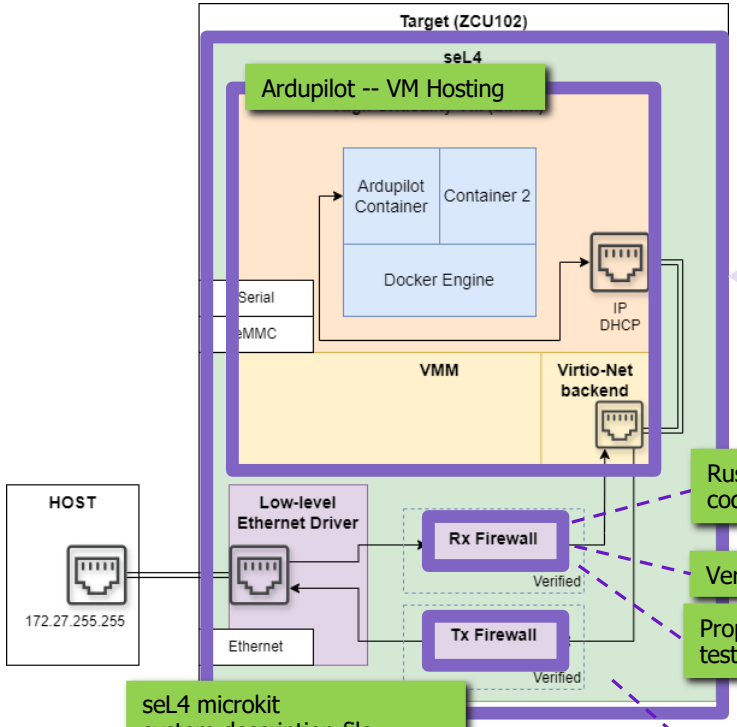
"Verifying Device Drivers with Pancake"

Robert Vanvossen Talk

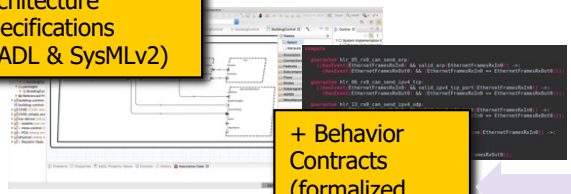
Dornerworks

"Rust-based drivers and verified rust applications on seL4" -- *Thursday*

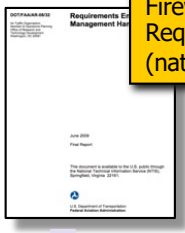
INSPECTA – Public Demonstrator Example



Architecture Specifications (AADL & SysMLv2)



+ Behavior Contracts (formalized Firewall Rules)



Firewall Requirements (natural language)

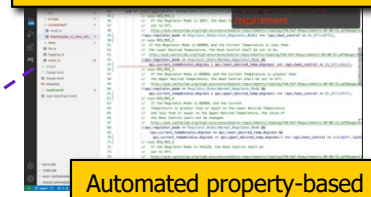
HAMR Generation

Recent demo!

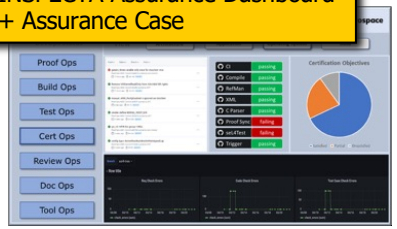
Automated..

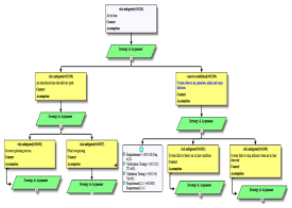
- metrics generation
- traceability info
- attestation (w/ KU)
- assurance case evidence

Verus Verification that Firewall implementation conforms to Requirements



INSPECTA Assurance Dashboard + Assurance Case





Rust application code skeletons

Verus Contracts

Property—based testing infrastructure

Automated property-based testing that Firewall implementation conforms to Requirements

seL4 microkit system description file (partitioning specification)

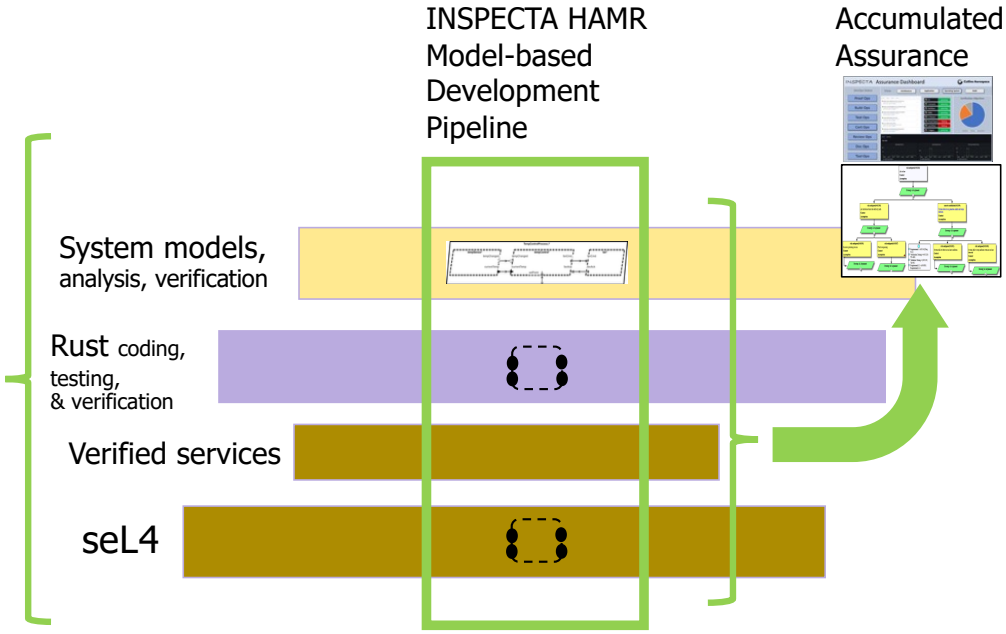
AADL-compliant threading and communication implemented using microkit primities

HAMR - SysMLv2/AADL to Rust + seL4

...Dornerworks found a 10x reduction in development time.

INSPECTA "PROVERS Pipeline" Scope

A primary goal of PROVERS is to demonstrate "pipelines" of formal methods capabilities. Designing and managing the INSPECTA "pipeline" entails a lot of extra work…



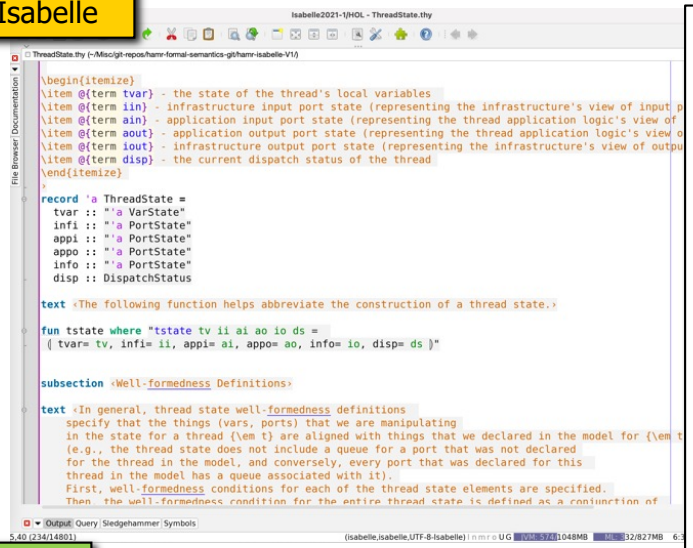
To fully demonstrate pipeline concepts within program timeline, the scope of the pipeline needs to be narrower than that of the individual technologies

- Interactions across the pipelines stages are organized a **core** set of computational and data **abstractions** that are amenable to **formal verification**
- **Semantics** of these abstractions must be maintained and **traceable across the stages**
- Claims, contributions of stages, and assurance evidence must be accumulated across the stages

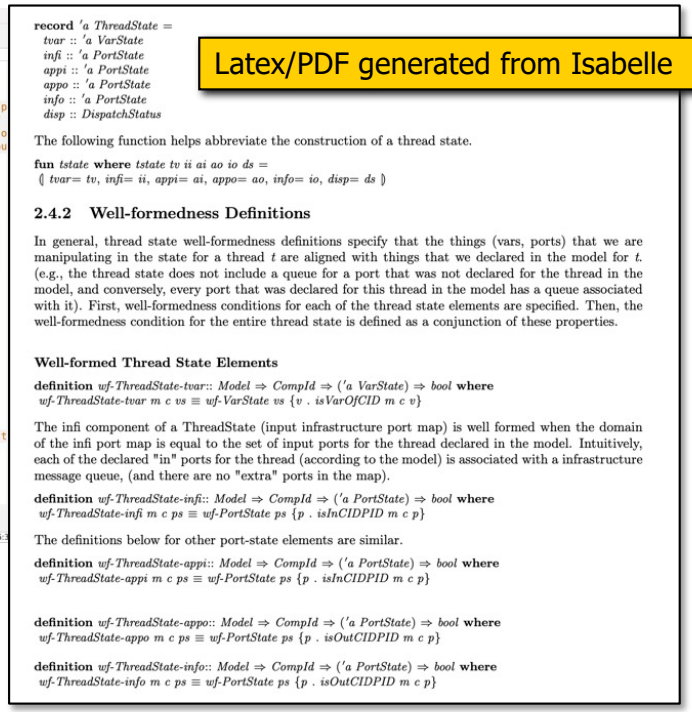
HAMR Formal Semantics for INSPECTA Pipeline

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

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*}

Joint work with Stefan Hallerstede (U. Aarhus)

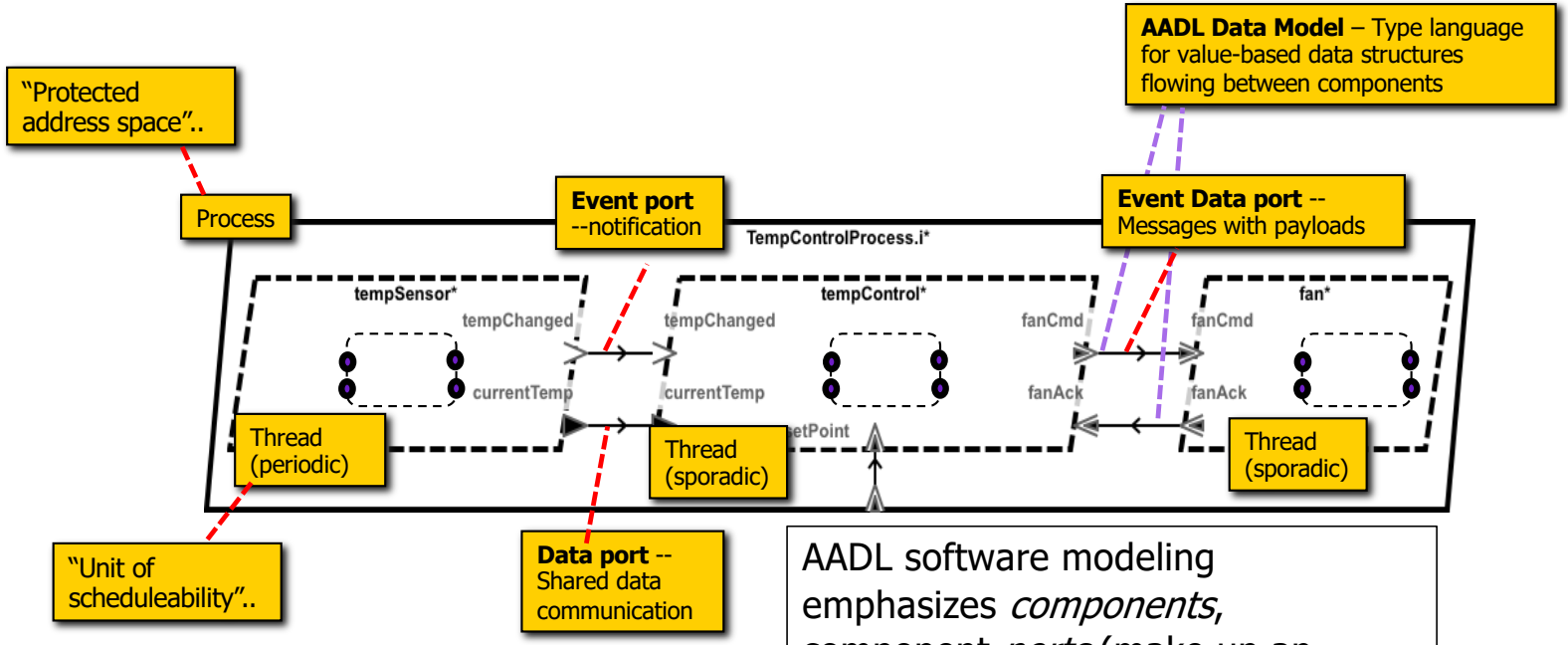
PROVERS

- 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

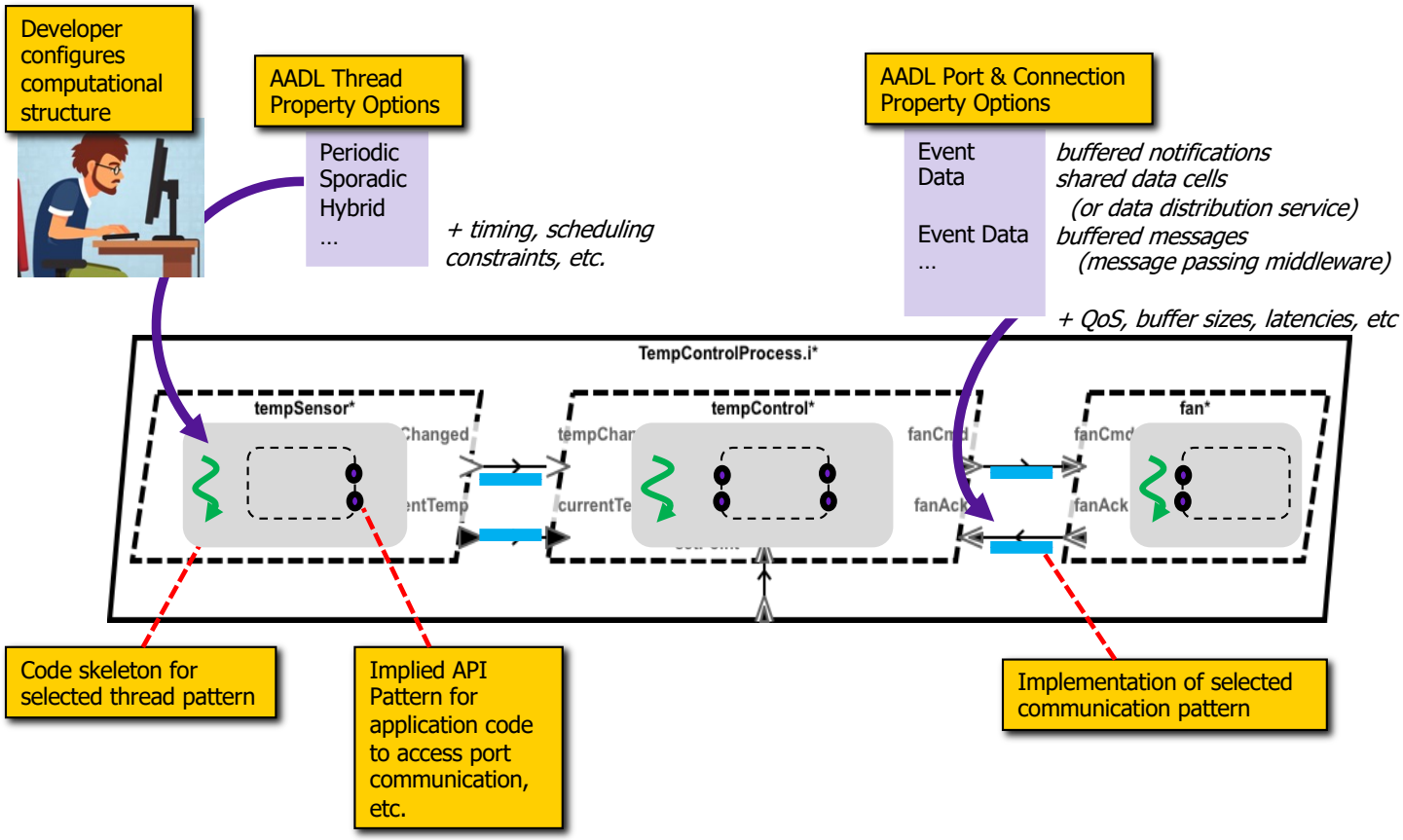
AADL Modeling Concepts

Each AADL modeling element is classified according to its role in embedded system architecture…

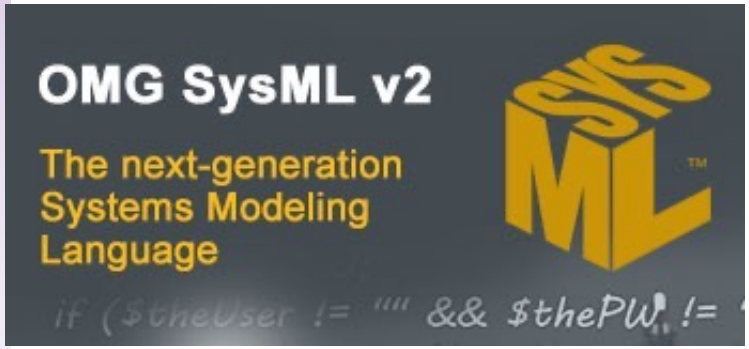


AADL software modeling emphasizes *components*, component *ports* (make up an interface), and communication *connections* between ports

AADL Modeling Concepts



AADL to SysMLv2



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

- 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
- Re-engineered from the ground up
  - No backwards compatibility with SysMLv1 except through translation
  - Not built as a profile of UML
- Will have wide-ranging commercial tool support as well as open source implementations

Standardization Effort - Migrating AADL to SysMLv2

OMG

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)

4 AADLv2 / SysMLv2 review RTESC WG

- Working with OMG RTESC working group to prototype AADL concepts in SysMLv2
- We are one of the most active participants working on building end-to-end tools for formal methods and code generation
- "Trail blazers" on integrating formal contract languages in SysMLv2 IDE

VSCode SysMLv2 HAMR Front End

We developed a VSCode SysMLv2 HAMR front-end based on the SysIDE VSCode plug-in

SysMLv2 component interfaces

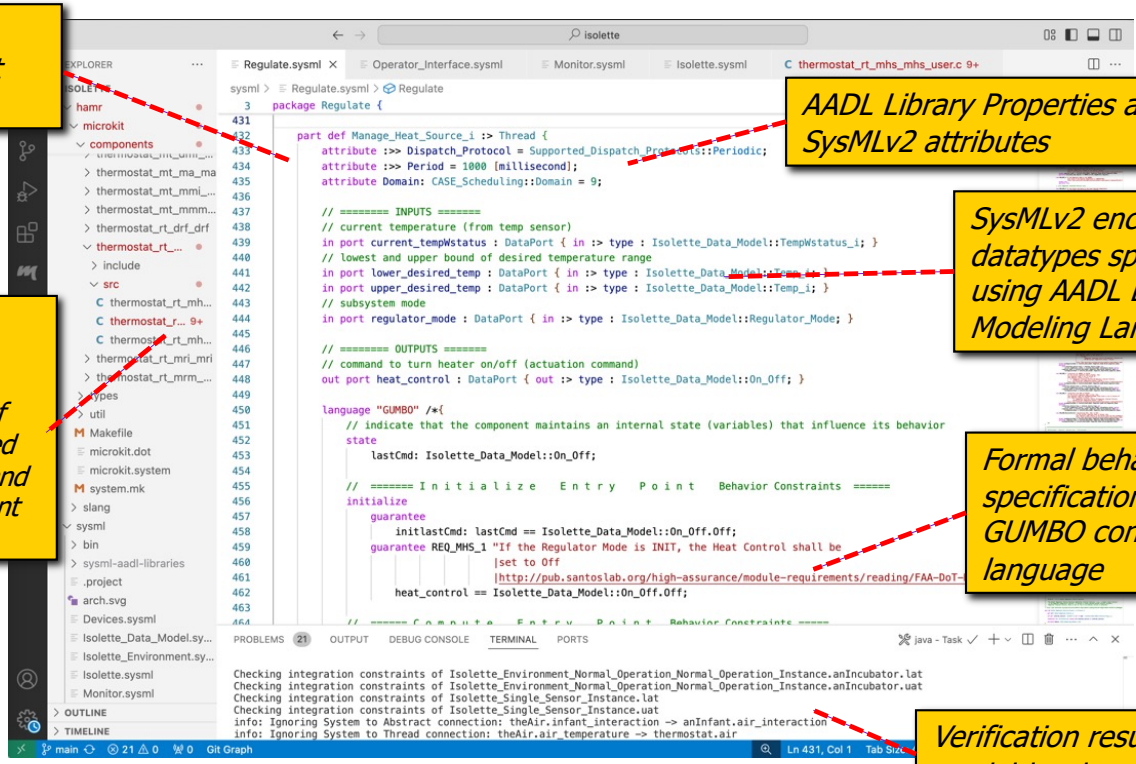
Code-level artifacts in same IDE: *Integration of MicroKit-based Slang, Rust and C development for seL4*

AADL Library Properties as SysMLv2 attributes

SysMLv2 encodings of datatypes specified using AADL Data Modeling Language

Formal behavior specifications in GUMBO contract language

Verification results for model-level contracts



Artifacts & Workflow -- Detailed Technical Report

Isolette – Infant Incubator

- *9 Real-time Tasks*
- *~40 component-level requirements*
- *Interestesting modal behavior*

End-to-end Artifacts

- ConOps
- Use Cases
- Requirements
- Models
- Contracts
- Testing
- Verification
- Assurance Case

HAMR - SysMLv2/AADL to Rust + seL4

The Isolette System: Illustrating End-to-End Artifacts for Rigorous Model-based Engineering

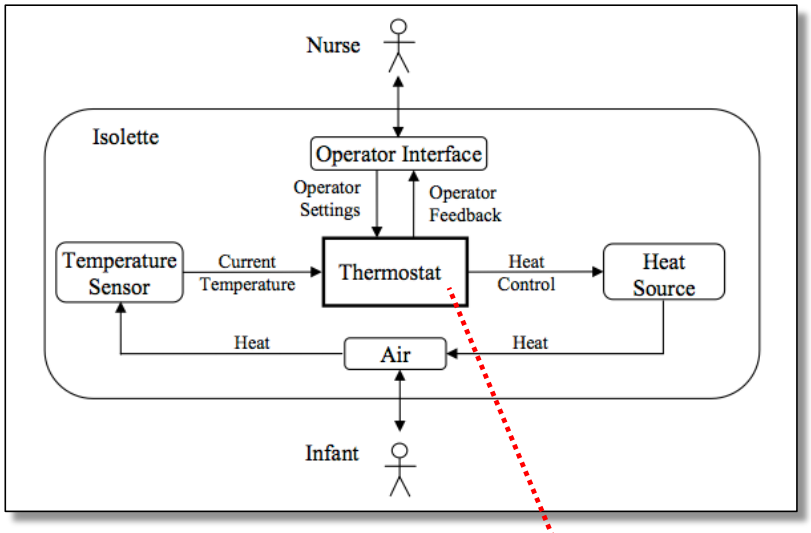
(Collins Aerospace INSPECTA Technical Report)

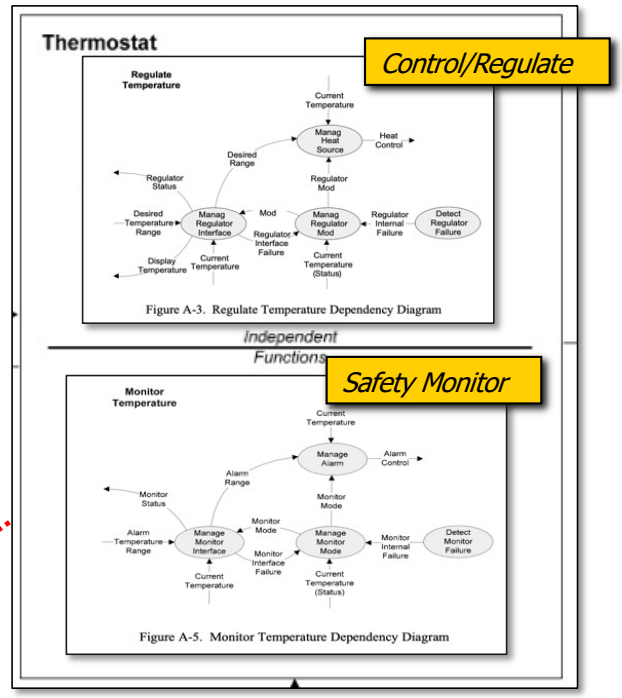
John Hatcliff and Jason Belt

50+ page report w/ Git repo and videos

REMH - Informal Designs

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

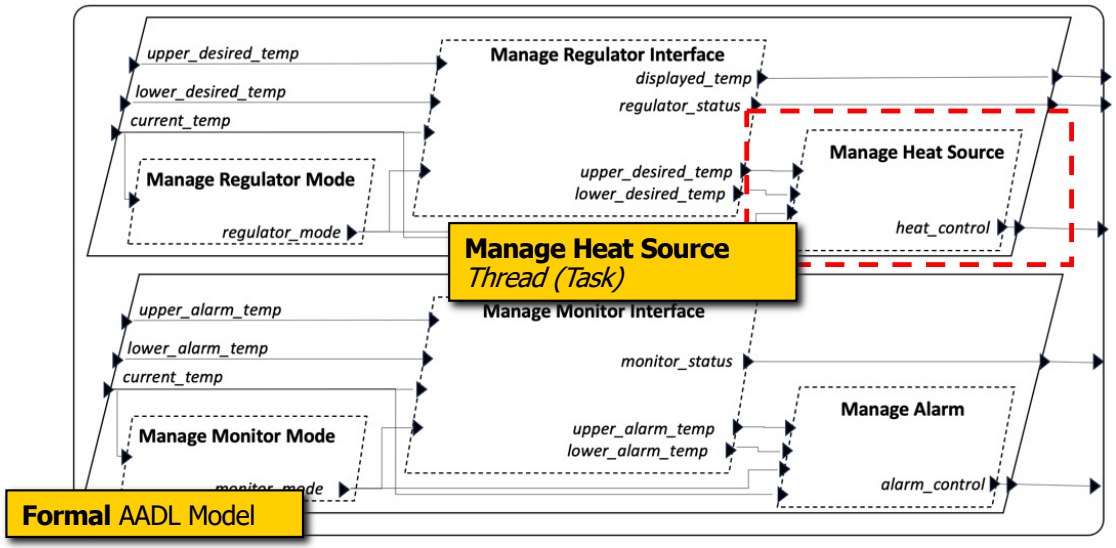


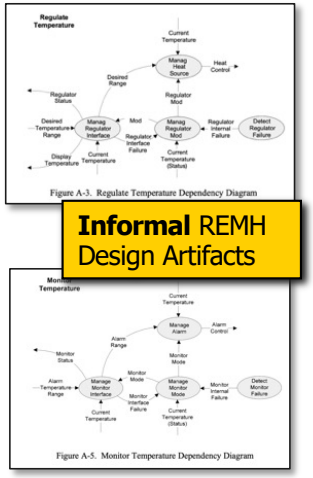


Thermostat decomposed into Regulate Temperature and Monitor Temperature functions.

Using AADL to Represent Design

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

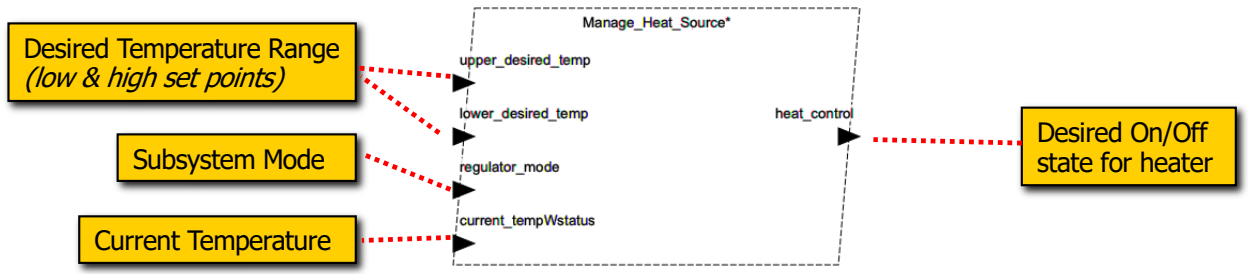




*This example (software aspects) is worked **completely end-to-end** from requirements, to contracts, to automatically tested and verified application code, to deployment on seL4, Linux, JVM, JavaScript.*
All artifacts are publicly available.

Manage Heat Source Thread

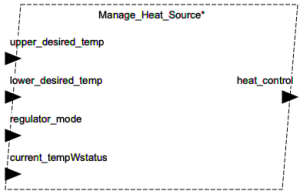
AADL Interface for **Manage Heat Source** Thread



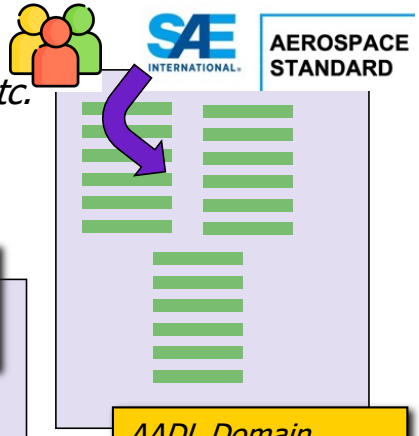

```
thread Manage_Heat_Source
  features
    -- ======== INPUTS =======
    -- ("Current Temperature") - current temperature (from temp sensor)
    current_tempWstatus: in data port Isolette_Data_Model::TempWstatus.impl;
    -- ("Desired Range") - lowest and upper bound of desired temperature range
    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 + AADL Modeling Concepts



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



AADL Domain Library for SysMLv2

Mark as AADL **`thread`**

Mark ports with AADL port categories and data types

SysMLv2

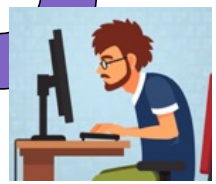
```
part def Manage_Heat_Source_i :> Thread  {

  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;
}
```

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 Side-by-Side

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;
```

Appearance is similar

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;
```

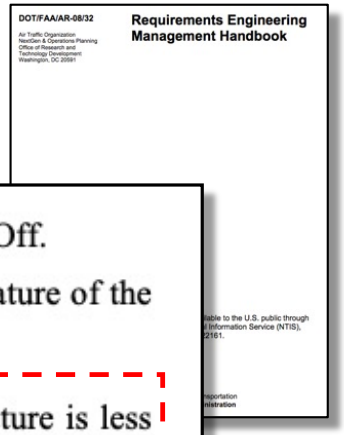
Challenges

Challenges in migrating AADL Formal Methods to SysMLv2

- SysMLv2 has no "annex mechanism"; need to figure out how to represent AADL Annexes
  - behavior contracts, architectural constraints language, hazard analysis
- Representation of AADL Properties
  - model configuration parameters
- Formal semantics of run-time behavior
  - Development of SysMLv2 "semantics" and "formal methods" is spread across several OMG working groups and is struggling to focus
  - SysMLv2 is big and general, so it is hard for committees to develop a precise semantics that satisfies their committee mandate

Natural Language Requirements for Thread

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



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.

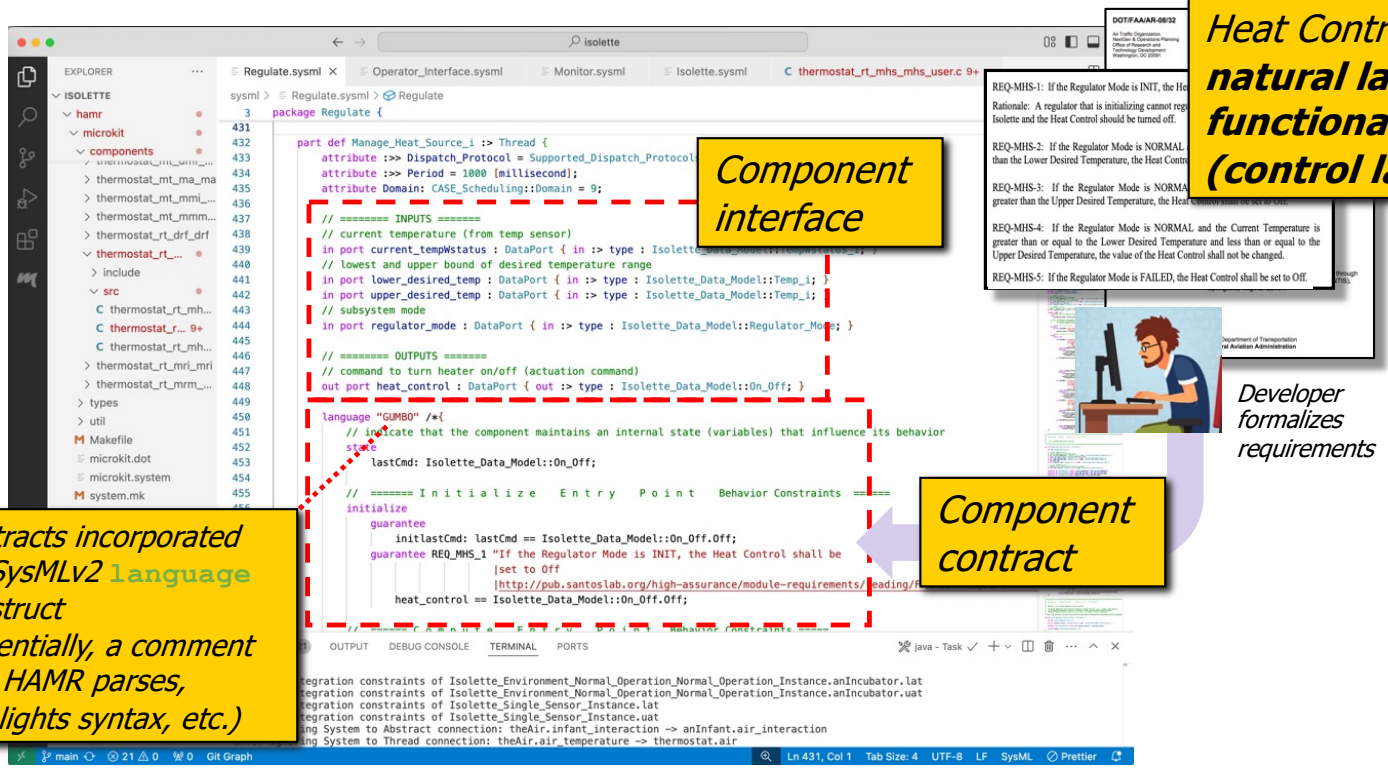
REQ-MHS-3: If the Regulator Mode is NORMAL and the Current Temperature is greater than the Upper Desired Temperature, the Heat Control shall be set to Off.

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.

Component Requirements to Contracts

GUMBO contracts are written together with the thread interface in the VSCode SysIDE plug-in **using a customized editor extension that we developed to support contracts**



Component interface

Heat Controller Task ***natural language functional requirements (control laws)***

Developer formalizes requirements

Component contract

Contracts incorporated via SysMLv2 `language` *construct (essentially, a comment that HAMR parses, highlights syntax, etc.)*

Component Requirements to Contracts

Example: One contract from *heater control laws* in **Manage Heat Source** Thread (a periodic component), with traceability to natural language requirements.

Mode condition
(..if the mode is Normal)

Compare current temperature to desired range
(..if temperature is below the target range)

...

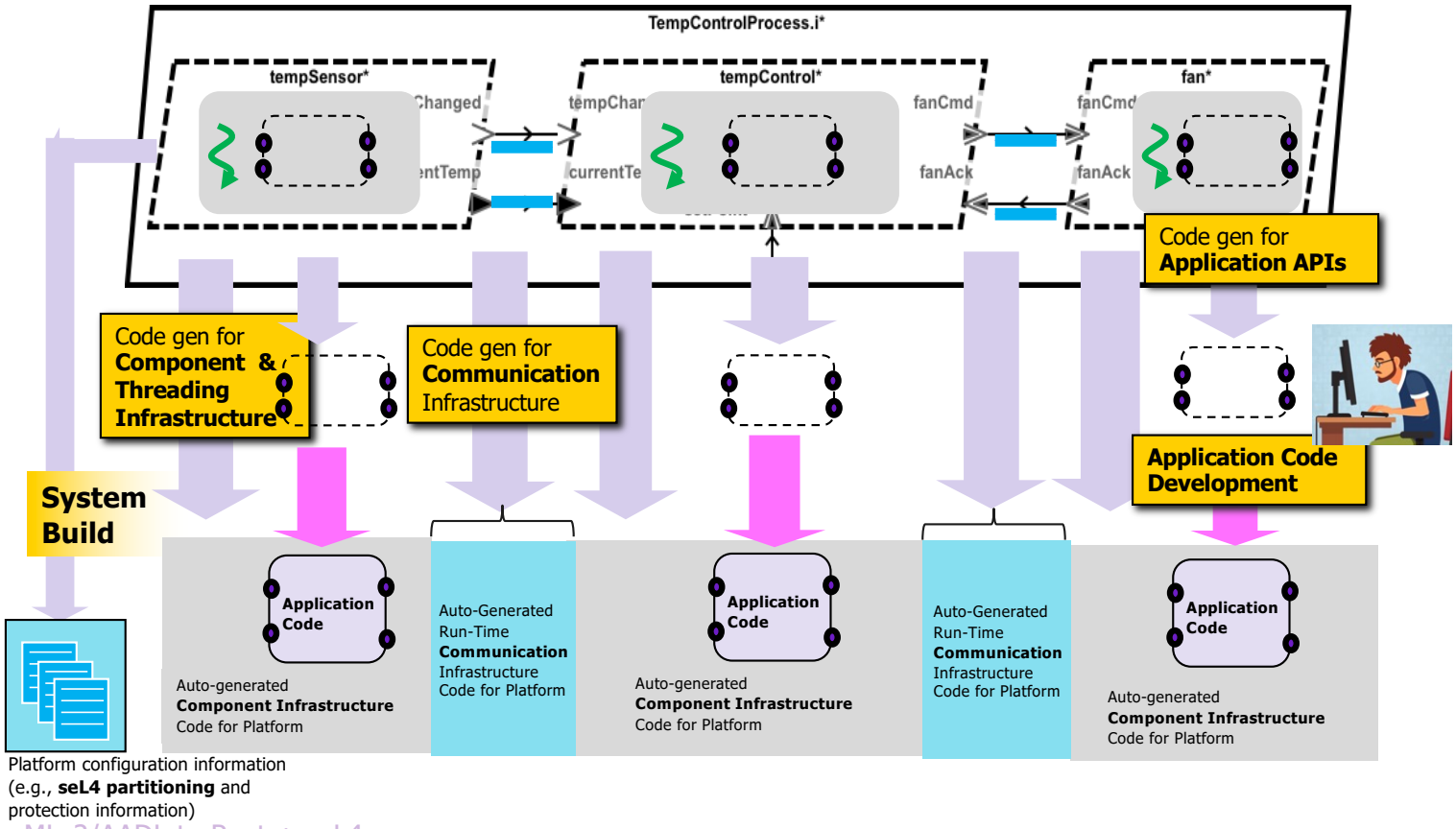
```
case REQ_MHS_2 "If the Regulator Mode is NORMAL and the Current Temperature is less than
        |the Lower Desired Temperature, the Heat Control shall be set to On.":
  assume (regulator_mode == Isolette_Data_Model::Regulator_Mode.Normal_Regulator_Mode)
    & (current_tempWstatus.value < lower_desired_temp.value);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

...

Set the desired state of the heater
(…turn heater On, to warm up the Isolette)

OSATE AADL Editor

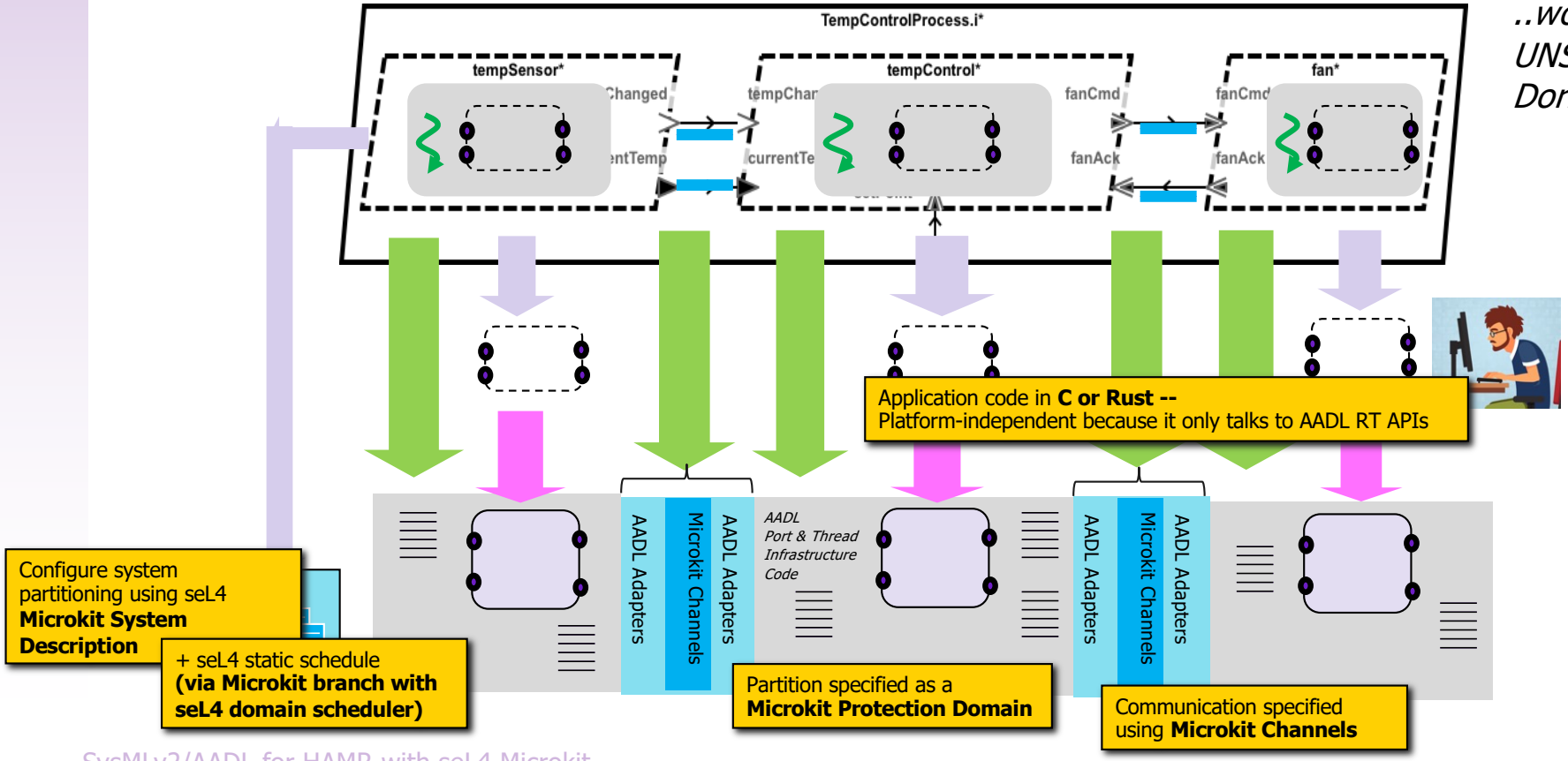
HAMR Code Generation



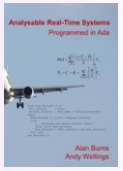
HAMR Code Generation

For seL4, the process is instantiated like this…

..working with UNSW and DornerWorks

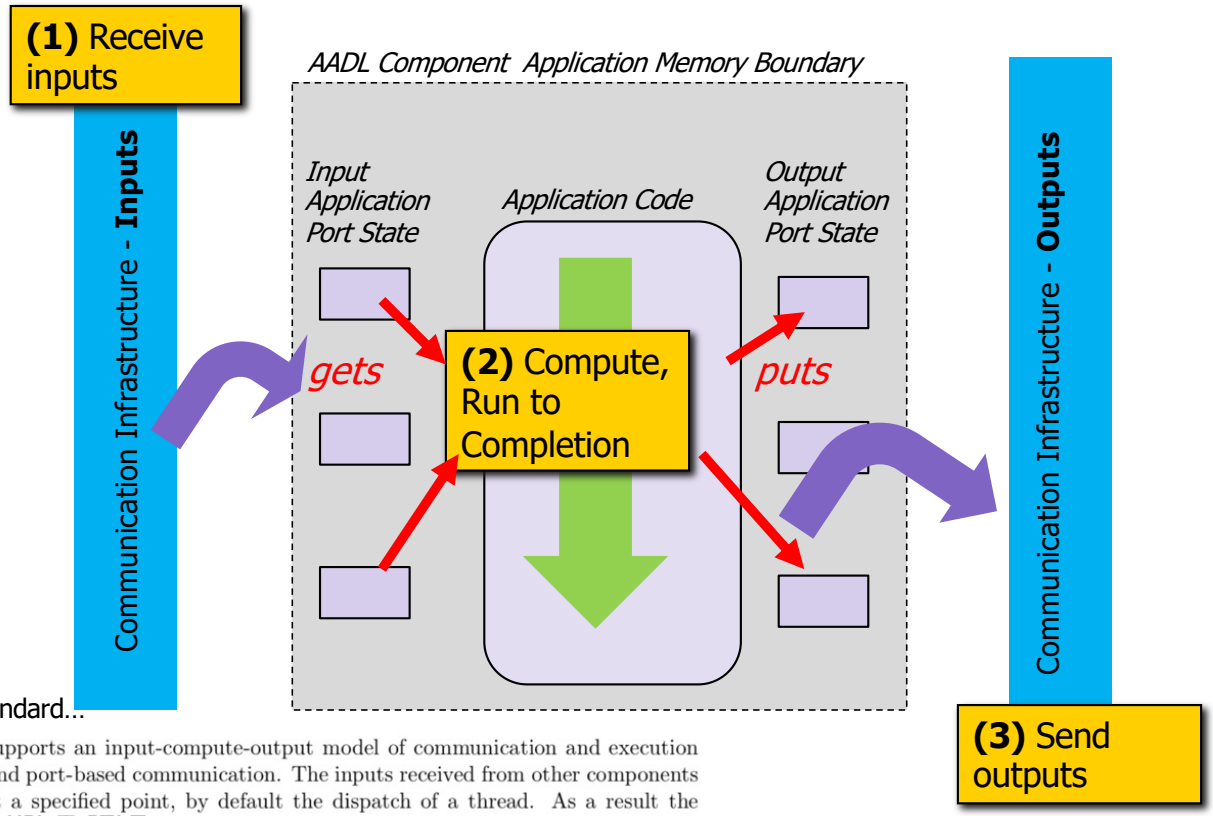


AADL Port and Thread Execution Semantics



"Analyzeable Real-Time Systems"
Burns & Wellings

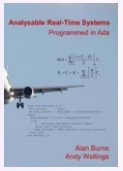
On each dispatch, AADL threads follow a well-known **input-compute-output** pattern for real-time tasks that simplifies analysis and verification…



From AADL standard…

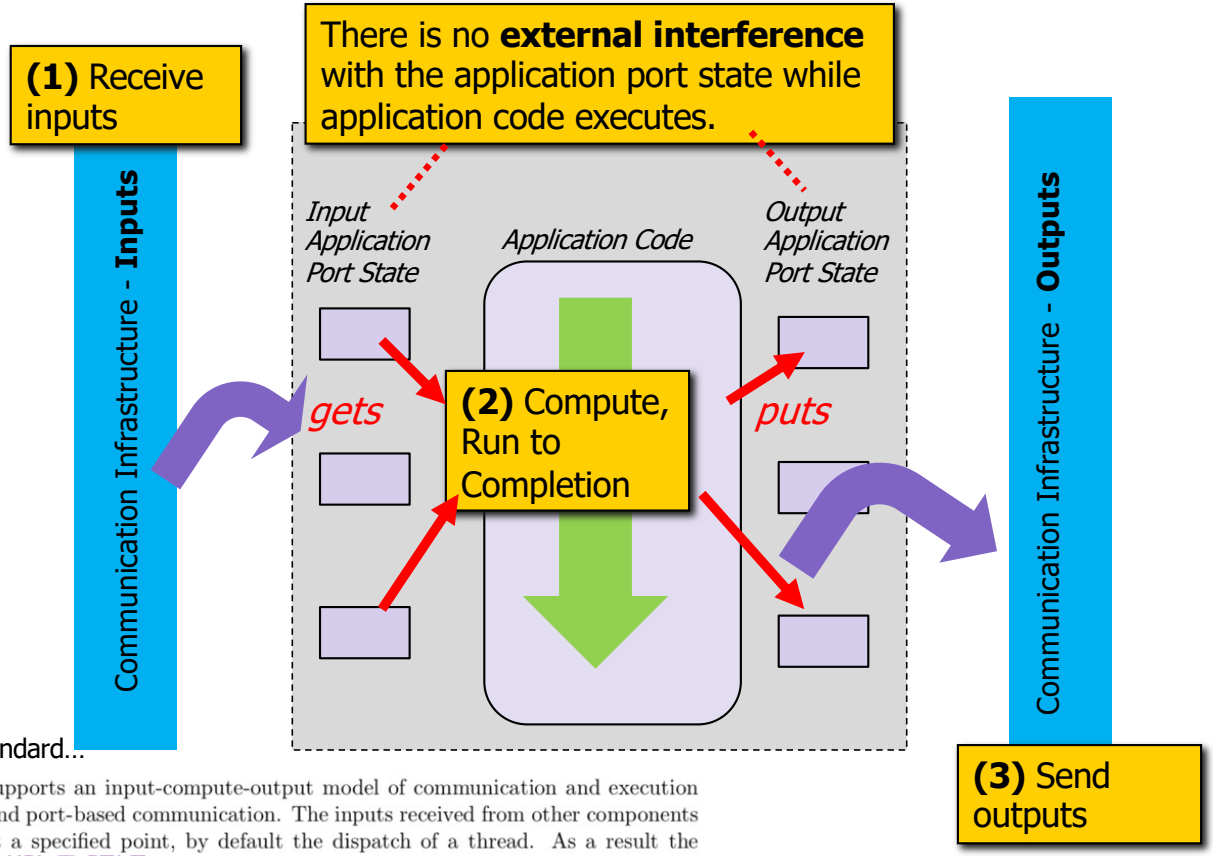
(2) AADL supports an input-compute-output model of communication and execution for threads and port-based communication. The inputs received from other components are frozen at a specified point, by default the dispatch of a thread. As a result the

AADL Port and Thread Execution Semantics



"Analyzeable Real-Time Systems"
Burns & Wellings

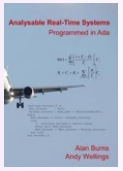
AADL tasking and port semantics ensures no interference with other threads or communication layer



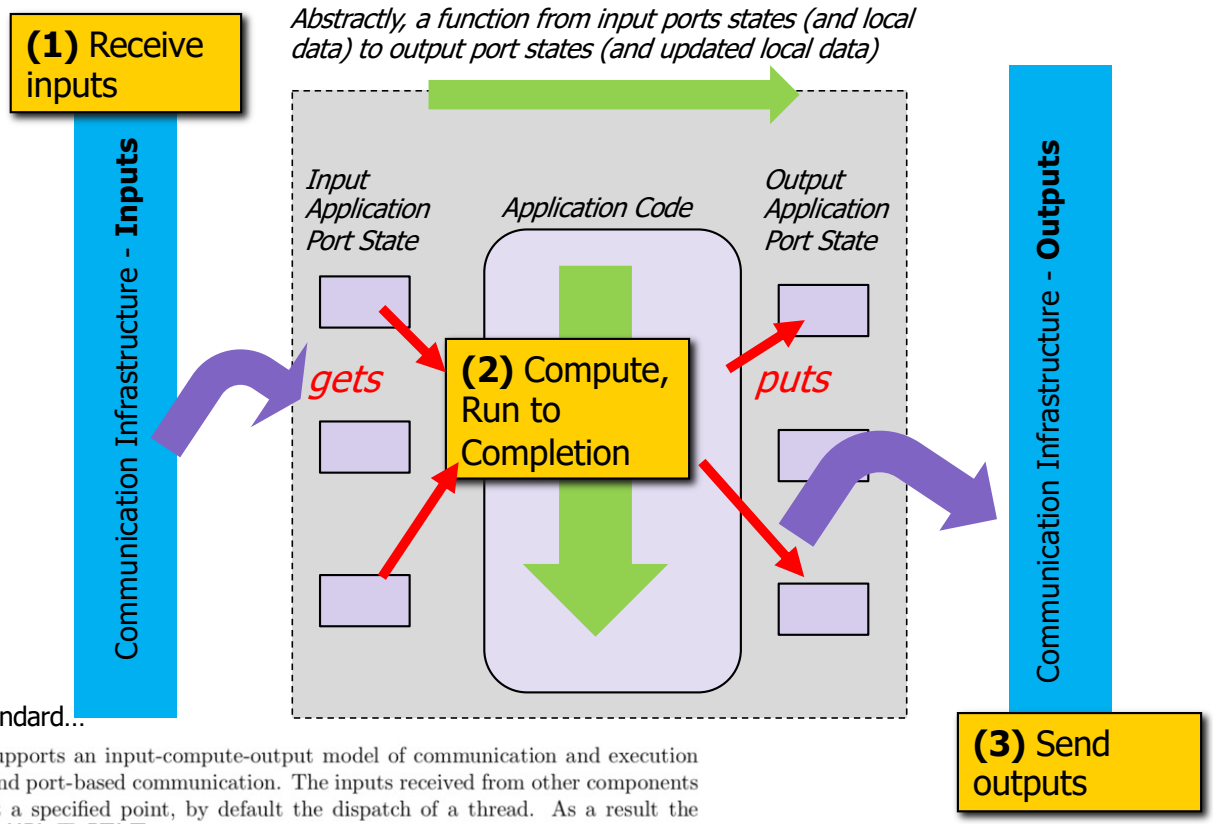
From AADL standard…

(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



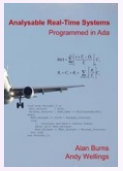
"Analyzeable Real-Time Systems"
Burns & Wellings



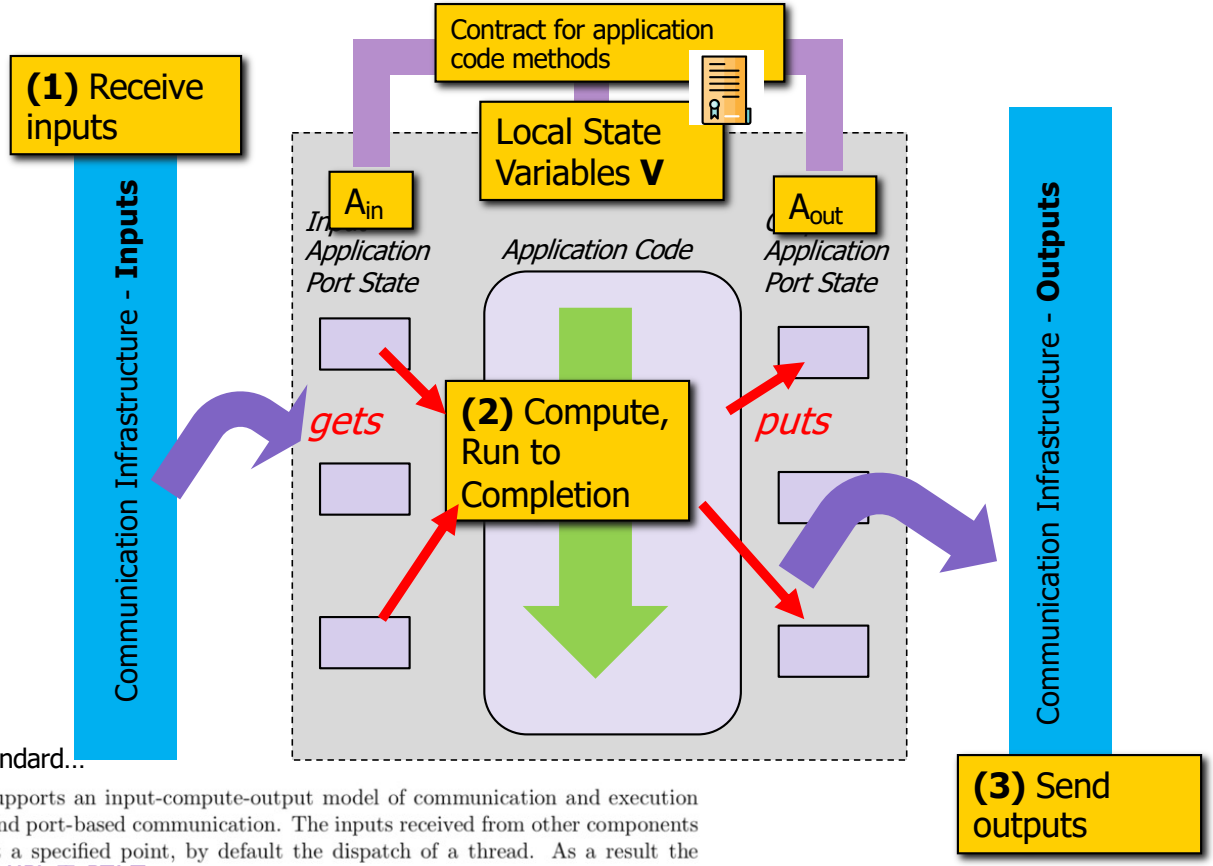
From AADL standard…

(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



"Analyzeable Real-Time Systems"
Burns & Wellings

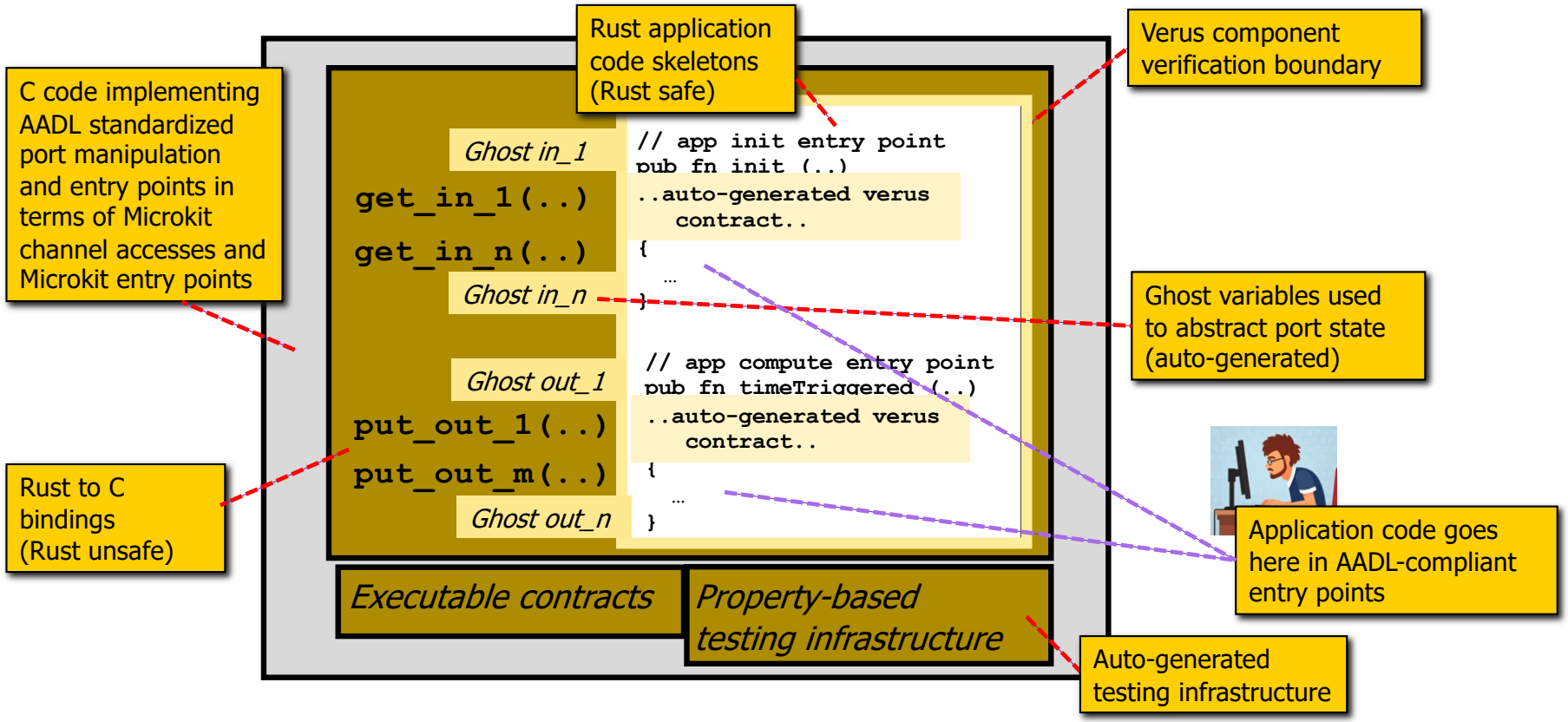


From AADL standard…

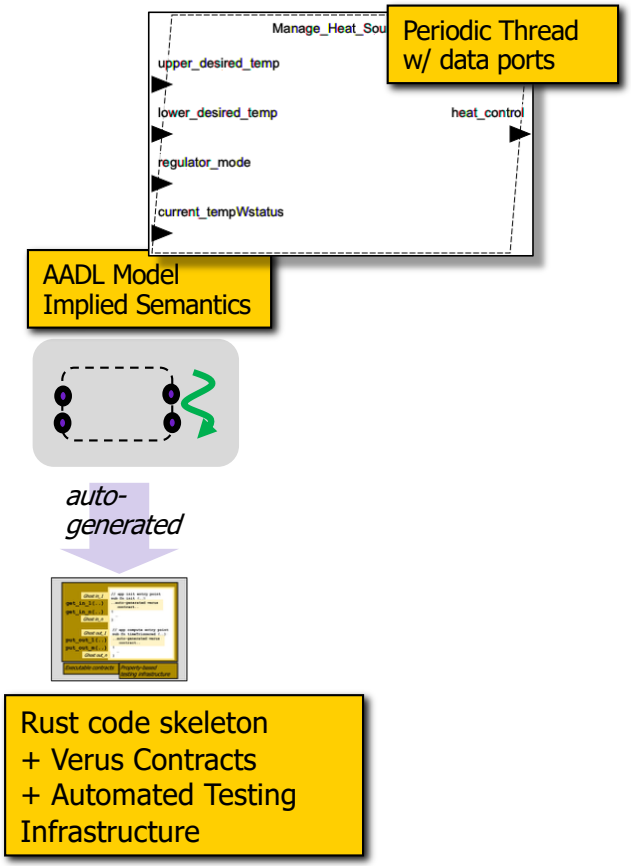
(2) AADL supports an input-compute-output model of communication and execution for threads and port-based communication. The inputs received from other components are frozen at a specified point, by default the dispatch of a thread. As a result the

Outline of Protection Domain Structure

For each SysMLv2/AADL *periodic* `Thread` component, HAMR generates the following Microkit PD code…



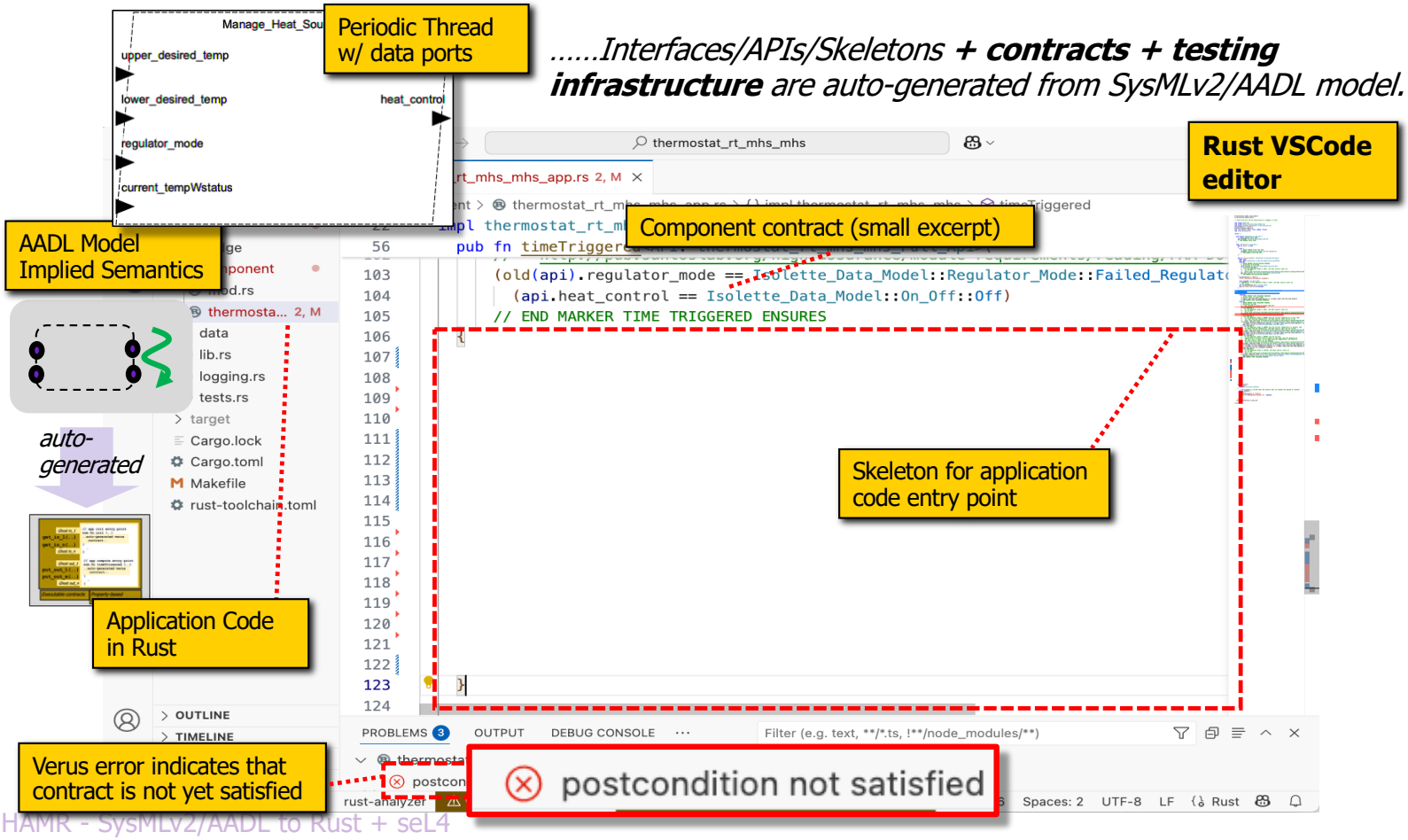
Auto-generated Skeleton, Contracts, Testing



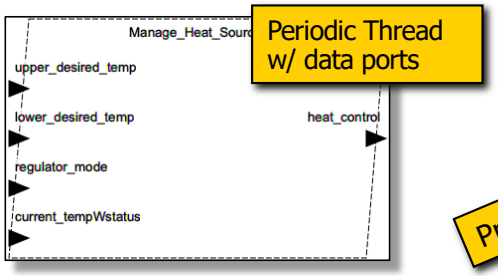
*……Interfaces/APIs/Skeletons **+ contracts + testing infrastructure** are auto-generated from SysMLv2/AADL model.*

Auto-generated Skeleton, Contracts, Testing

*.....Interfaces/APIs/Skeletons **+ contracts + testing infrastructure** are auto-generated from SysMLv2/AADL model.*



Verus Contract Auto-Generated From Model Contract



Periodic Thread w/ data ports

auto-generated

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

Pre

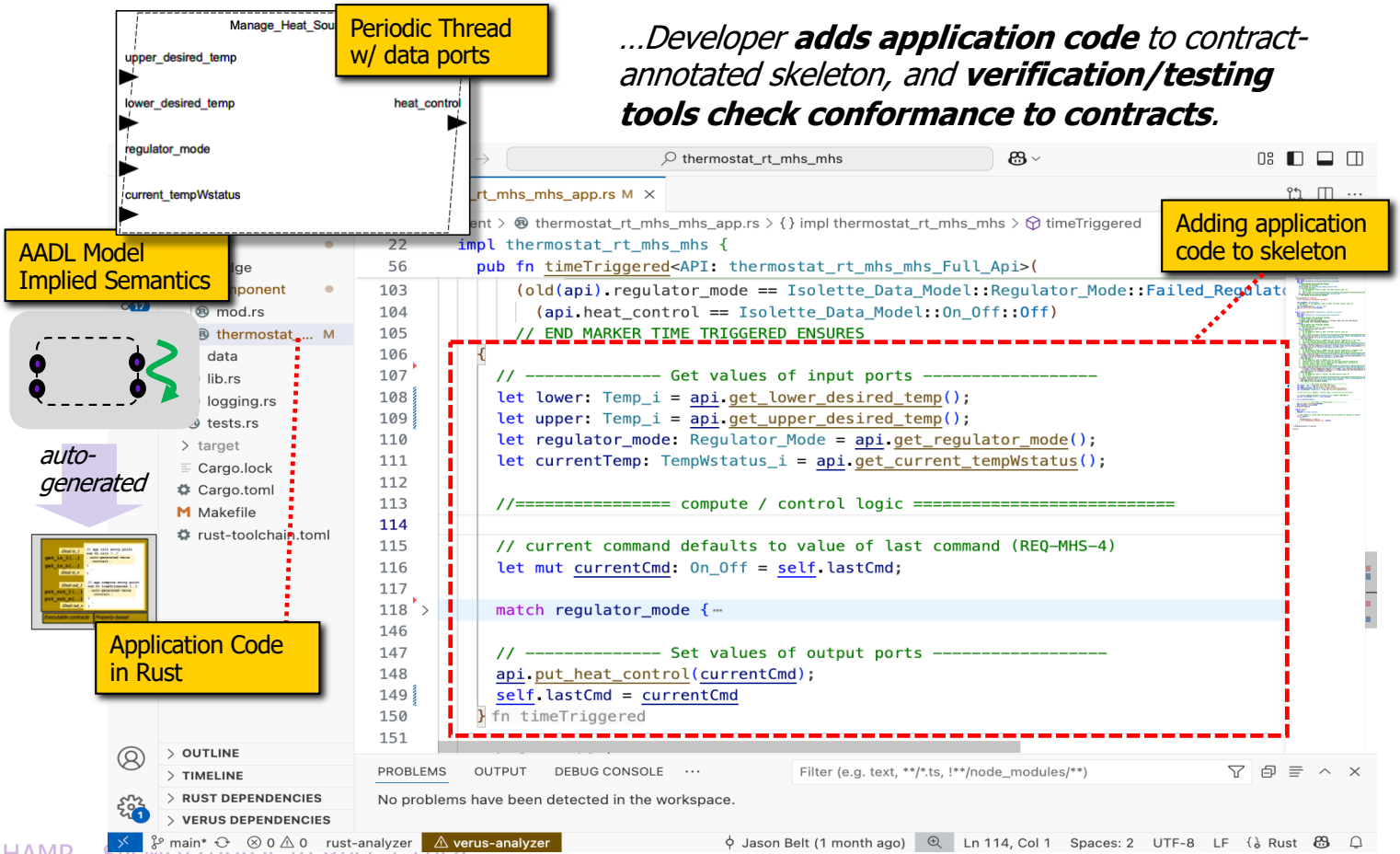
Post

```
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),
```

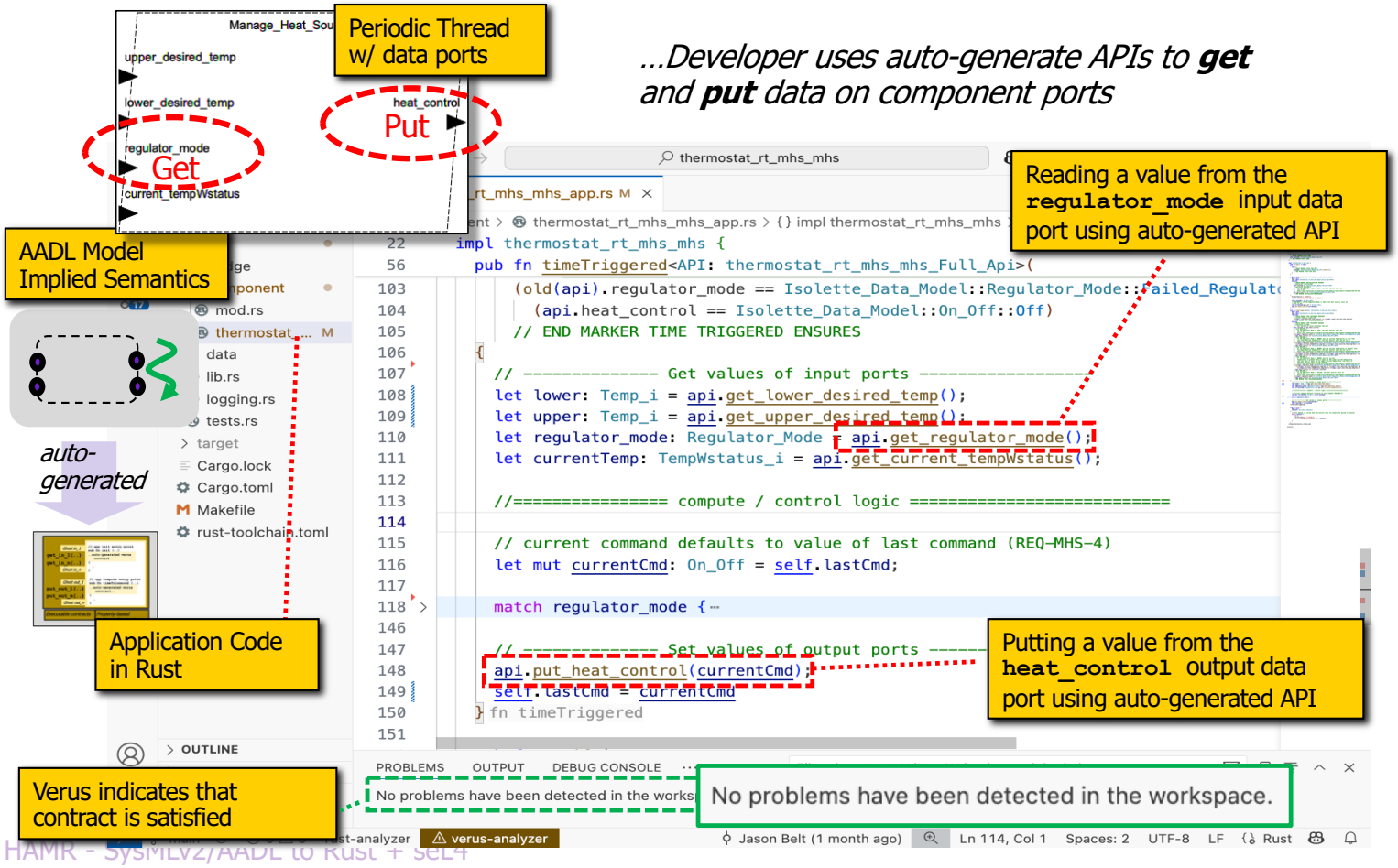
```
76      //   If the Regulator Mode is NORMAL and the Current Temperature is less than
77      //   the Lower Desired Temperature, the Heat Control shall be set to On.
78      ((old(api).regulator_mode == Regulator_Mode::Normal_Regulator_Mode) &&
79        (old(api).current_tempWstatus.degrees < old(api).lower_desired_temp.degrees))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

Coding and Background Verification



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

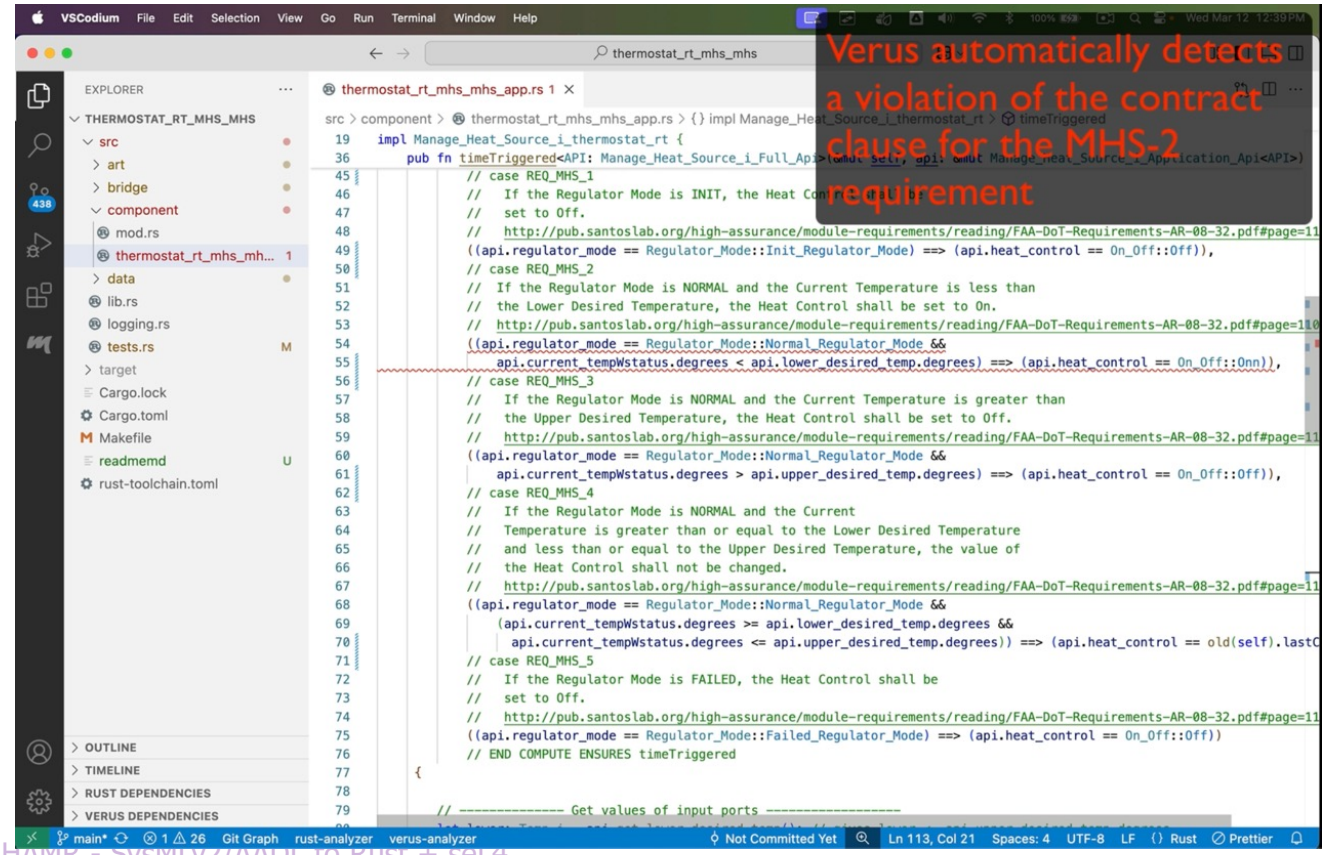
Coding and Background Verification



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

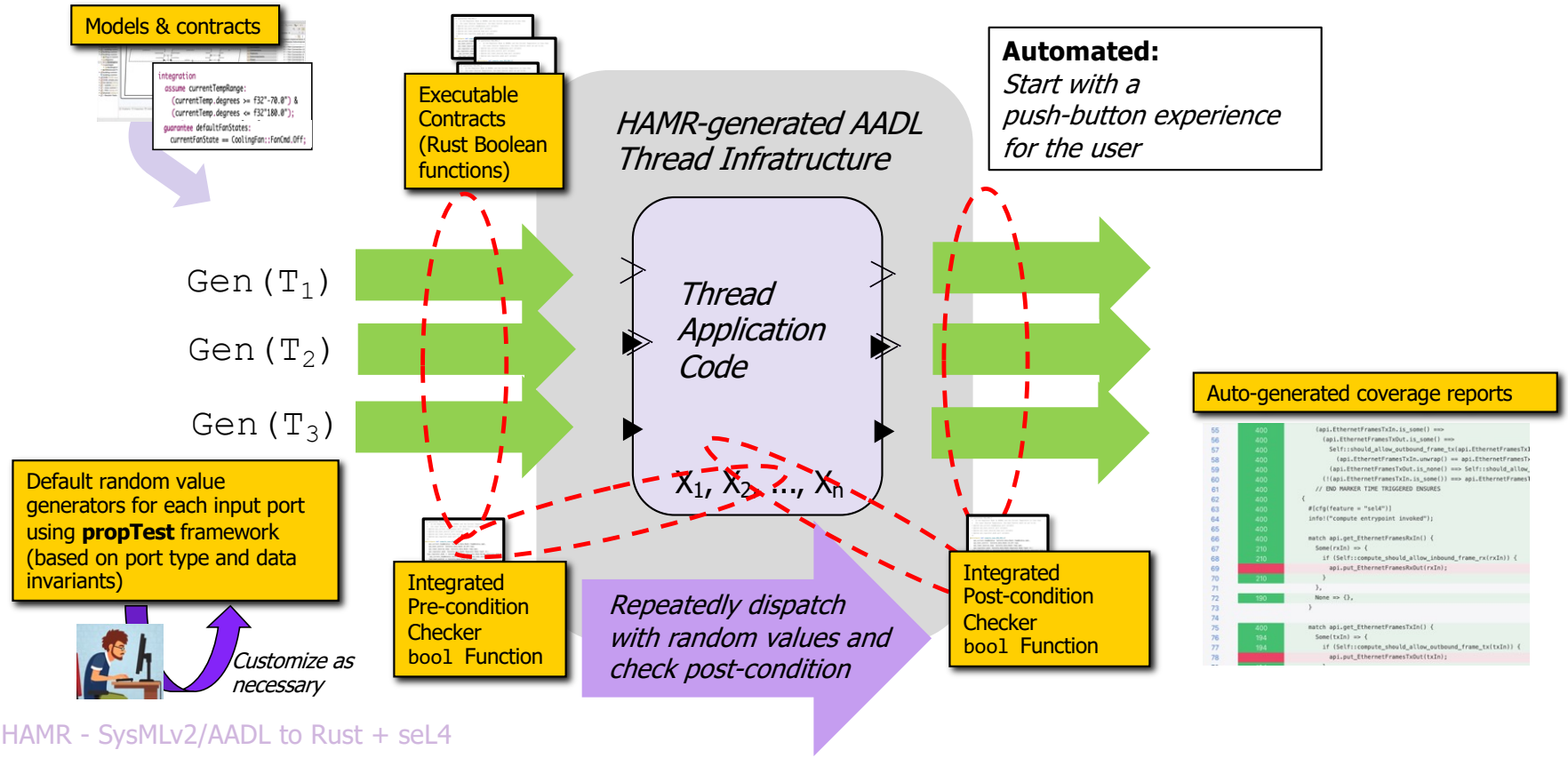
Demo

Verification of application code against contracts using Verus verification tool…



Automated Testing to Contracts

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.

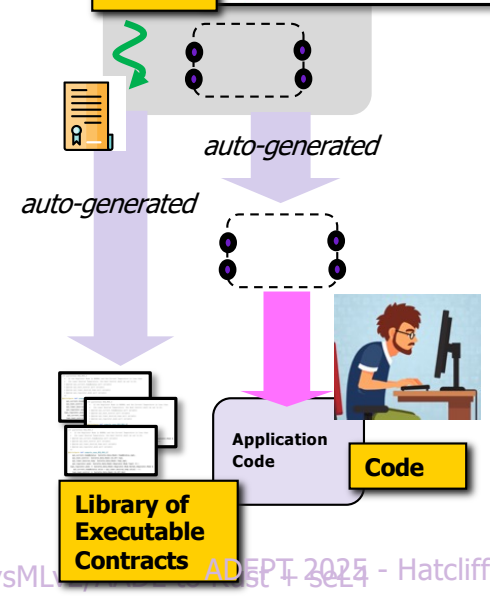


HAMR-generated *Executable* Contracts

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;
```



Rust Executable Contract (clause REQ_MHS_2)

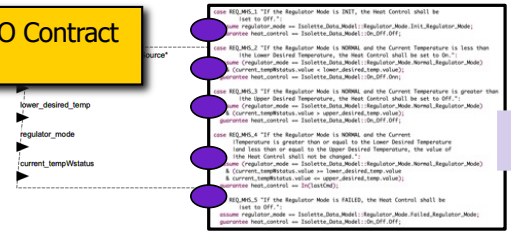
```
/** guarantee REQ_MHS_2
 *   If the Regulator Mode is NORMAL and the Current Temperature is less than
 *   the Lower Desired Temperature, the Heat Control shall be set to On.
 * @param api_current_tempWstatus incoming data port
 * @param api_lower_desired_temp incoming data port
 * @param api_regulator_mode 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

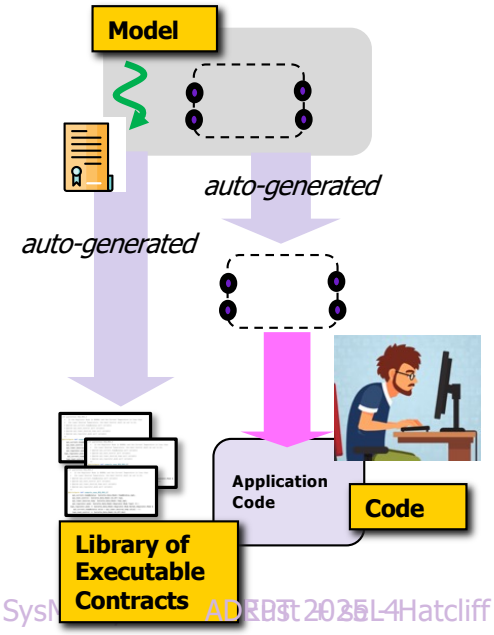
HAMR-generated *Executable* Contracts

Complete set of **Model-level** GUMBO contract clauses are translated to a hierarchy of executable Boolean functions in Rust (**code-level)** to form executable pre/post conditions and test oracles.

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.).*



Code: Auto-generated in Testing Library

```
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;
}
```

Rust Executable Contract (aggregated clauses for post-condition)

HAMR-generated Randomizing Test Runner

HAMR automatically generates test runner infrastructure with default random value generators for each input port. The executable contract is automatically used behind the scenes as a test oracle.

```
testComputeCB_macro! {
  ▶ Run Test | Debug
  prop_testComputeCB_default, // test name
  config: ProptestConfig { // proptest configuration, built by overriding fields from default config
      cases: numValidComputeTestCases,
      max_global_rejects: numValidComputeTestCases * computeRejectRatio,
      verbose: verbosity,
      ..ProptestConfig::default()
  },
  // auto-generated strategies for generating each component input
  api_current_tempWstatus: test_api::Isolette_Data_Model_TempWstatus_i_strategy_default(),
  api_lower_desired_temp: test_api::Isolette_Data_Model_Temp_i_strategy_default(),
  api_regulator_mode: test_api::Isolette_Data_Model_Regulator_Mode_strategy_default(),
  api_upper_desired_temp: test_api::Isolette_Data_Model_Temp_i_strategy_default()
}
```

Press this button and you automatically get 1000's of random tests against the component contract

auto-generated configurations for propTest framework

auto-generated propTest random value generators for each input port

HAMR-generated Randomizing Test Runner

Default random generators are often easy to customize to increase coverage, reduce #'s of discarded tests, obtain tests for specific features, etc.

```
testComputeCB_macro! {
  ▶ Run Test | Debug
  prop_testComputeCB_default, // test name
  config: ProptestConfig { // proptest configuration, built by overriding fields from default config
      cases: numValidComputeTestCases,
      max_global_rejects: numValidComputeTestCases * computeRejectRatio,
      verbose: verbosity,
      ..ProptestConfig::default()
  },
```

Customizing a numeric generator to a particular range

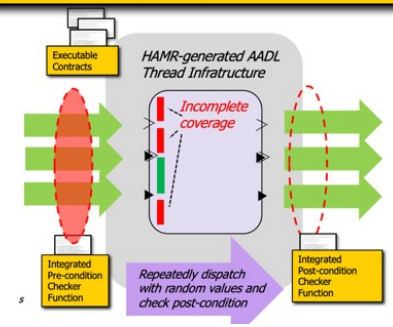
```
// developer-customized strategies for generating each component input
api_current_tempWstatus: test_api::Isolette_Data_Model_TempWstatus_i_stategy_cust(
    95..=103,
    test_api::Isolette_Data_Model_ValueStatus_strategy_default()),
api_lower_desired_temp: test_api::Isolette_Data_Model_Temp_i_stategy_cust(94..=105),
api_regulator_mode: test_api::Isolette_Data_Model_Regulator_Mode_strategy_default(),
api_upper_desired_temp: test_api::Isolette_Data_Model_Temp_i_stategy_cust(94..=105)
```

Developer-customized generators

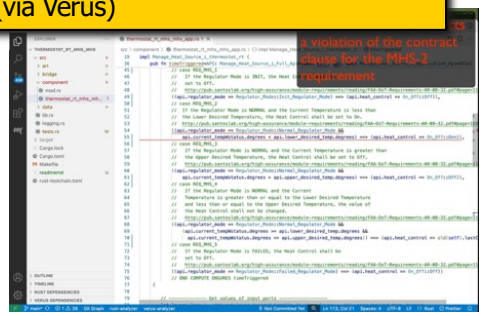
Benefits - Integrated Testing / Verification

- Immediately launch 1000's of default tests, check conformance to contracts
  - Debug contracts; gradually move to Verus
- When Verus verification fails, generate concrete failing tests that can be given to developers to run through debugger
- When Verus/SMT cannot handle certain language features; use testing for lower-confidence assurance
  - Maybe be a step taken before handing off certain VCs from Verus to Lean
- Testing and verification *derived from the exact same GUMBO contracts*

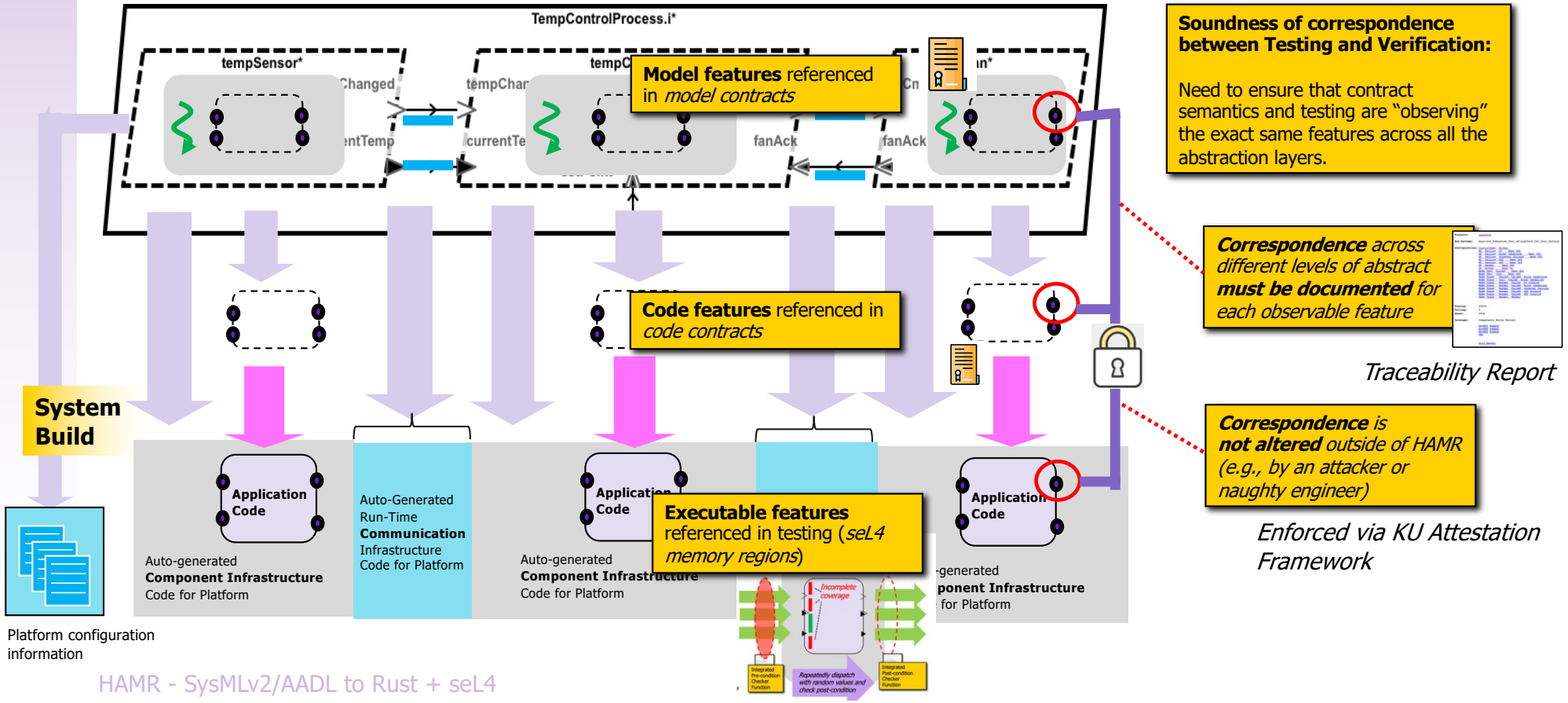
HAMR automated component testing (via Rust propTest)



HAMR automated verification (via Verus)

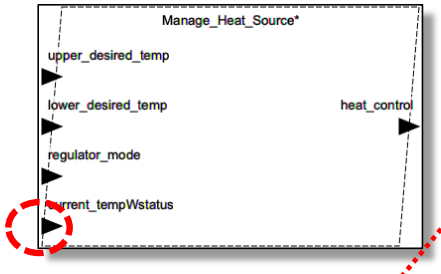


HAMR Observations/Traceability Framework



Auto-generated System Feature Traceability for Manage Heat Source Port (GitHub Markdown)

HAMR auto-generates traceability reports, e.g., for a port – relationships between **model, code, kernel** artifacts



(Model) SysMLv2 port declaration

```
_Source extends Abstract_Definitions::Periodic_Thread
    features
        -- ======== INPUTS =======
        -- ("Current Temperature") - curren
        current_tempWstatus: in data port I
```

(Code) HAMR-generated Rust API/implementation

```
Code  Blame  170 lines (152 loc) · 6.74 KB
    }
    pub fn get_current_tempWstatus(&mut self) -> (res : Isolette_Data_Model::TempWstatus_i)
      ensures
        old(self).current_tempWstatus == self.current_tempWstatus,
        res == self.current_tempWstatus,
        old(self).lower_desired_temp == self.lower_desired_temp,
        old(self).upper_desired_temp == self.upper_desired_temp,
        old(self).regulator_mode == self.regulator_mode,
        old(self).heat_control == self.heat_control
    {
      self.api.unverified_get_current_tempWstatus(&Ghost(self.current_tempWstatus))
    }
  }

  pub struct thermostat_rt_mhs_mhs_Initialization_Api;
  impl thermostat_rt_mhs_mhs_Api for thermostat_rt_mhs_mhs_Initialization_Api {}
  impl thermostat_rt_mhs_mhs_Put_Api for thermostat_rt_mhs_mhs_Initialization_Api {}
```

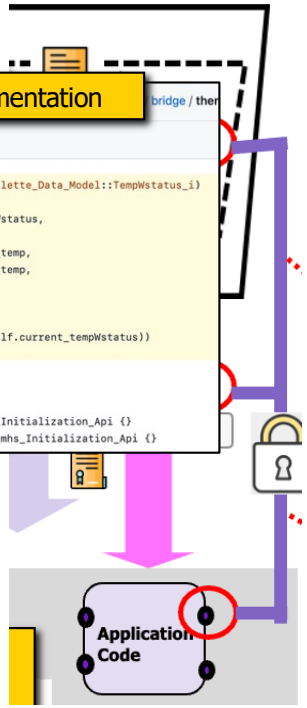
- **APIs**

Port Name	Direction	Kind	Payload	Realiza
current_tempWstatus	In	Data	Isolette_Data_Model::TempWstatus.i	Memory Map -> Rust API
lower_desired_temp	In	Data	Isolette_Data	
upper_desired_temp	In	Data	Isolette_Data	
regulator_mode	In	Data	Isolette_Data	
heat_control	Out	Data	Isolette_Data	

main INSPECTA-models / isolette / hamr / microkit / microkit.system

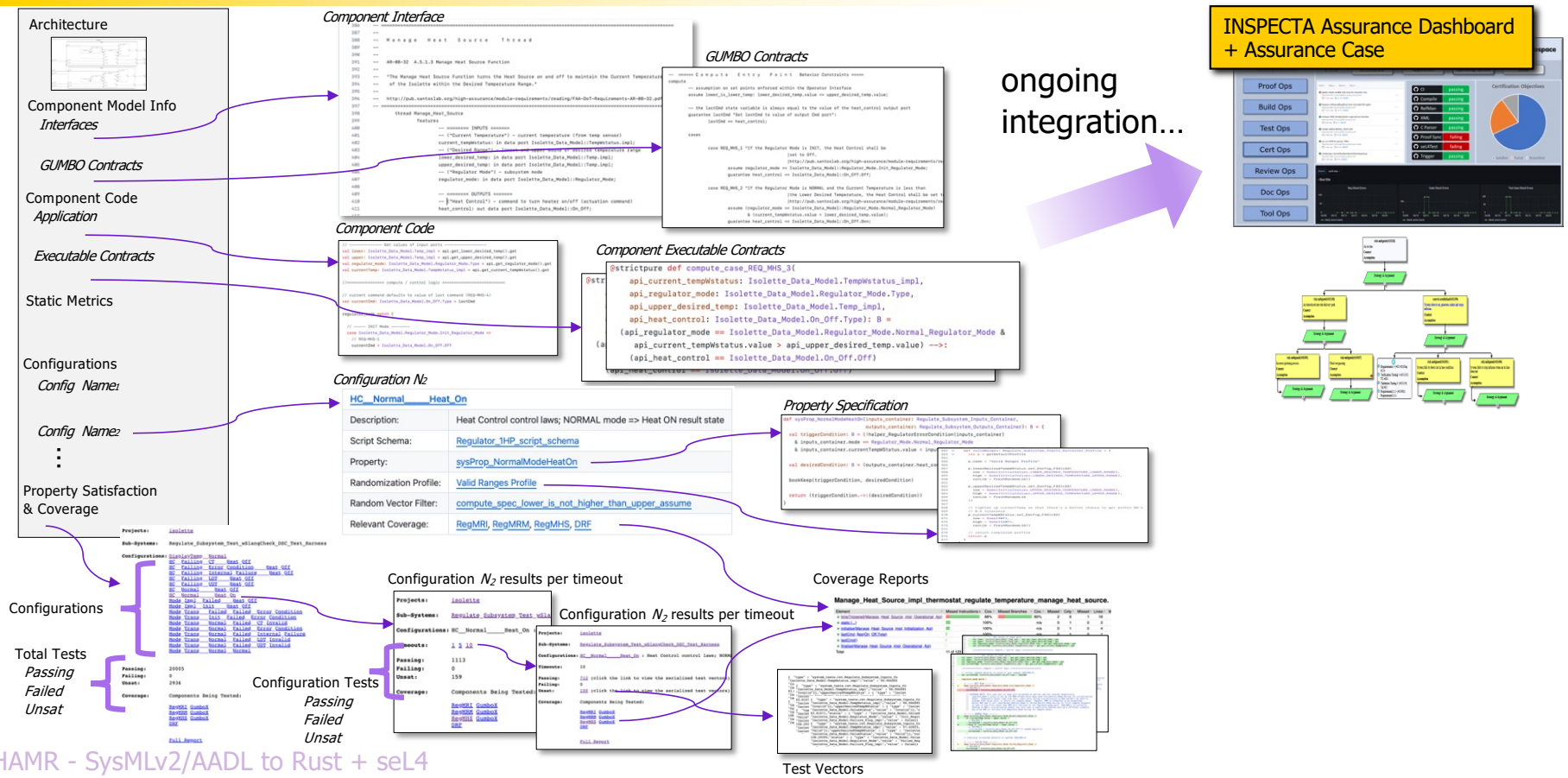
```
Code  Blame  530 lines (474 loc) · 21 KB
    />
    <map mr="Isolette_Single_Sensor_Instance_temperature_sensor_cpi_thermostat_
        vaddr="0x10_004_000"
        perms="r"
        setvar_vaddr="current_tempWstatus_queue_1"
    />
  </protection
</protection_d
```

(seL4 Microkernel) Declaration/protection of shared memory for port



Assurance and Traceability Reports

HAMR auto-generates a variety of assurance and traceability reports



Auto-generated Contract Traceability for Manage Heat Source Requirement (GitHub Markdown)

Contract Clause traceability

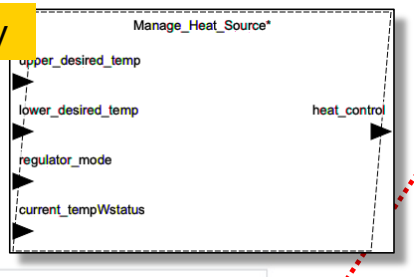
- GUMBO

State Variables		
lastCmd	GUMBO	Verus

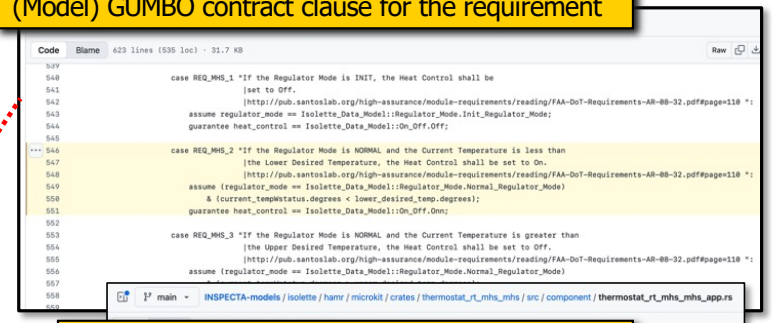
Initialize			
guarantee initlastCmd	GUMBO	Verus	GUMBOX
guarantee REQ_MHS_1	GUMBO	Verus	GUMBOX

Compute			
assume lower_is_lower_temp	GUMBO	Verus	GUMBOX
guarantee lastCmd	GUMBO	Verus	GUMBOX
case REQ_MHS_1	GUMBO	Verus	GUMBOX
case REQ_MHS_2	GUMBO	Verus	GUMBOX
case REQ_MHS_3	GUMBO	Verus	GUMBOX
case REQ_MHS_4	GUMBO	Verus	GUMBOX
case REQ_MHS_5	GUMBO	Verus	GUMBOX

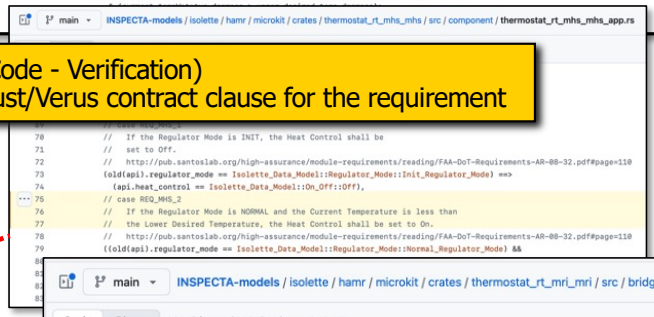
Req ID



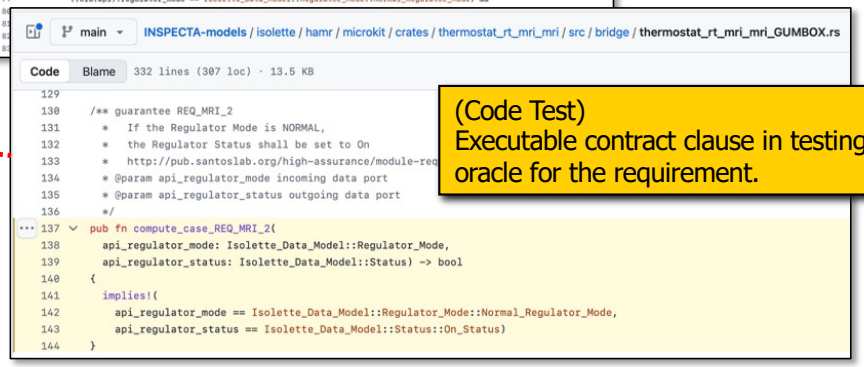
(Model) GUMBO contract clause for the requirement



(Code - Verification)
Rust/Verus contract clause for the requirement



(Code Test)
Executable contract clause in testing oracle for the requirement.



Conclusions -- Themes

- Raising the abstraction level
  - ”**patterns** for Microkit protection domains” **for application components**
  - choosing patterns to be amenable to component/system assurance
  - representing patterns/abstractions in **standardized modeling languages**
  - separating developer view of the pattern (higher-level) from seL4/microkit realization (lower-level)
- Auto-generation support for development tasks
  - build scripts, VM configuration, testing, logging
- Leveraging specifications for both verification and testing
- Integrating activities from broader industry ecosystems, especially those related to assurance activities

Plans - Next Six Months

- System Reasoning
  - Formal system specifications
  - System Testing / Run-time Monitoring
  - System Verification
- Continued evolution of Microkit target
  - More systematic scheduling and communication, support for LionsOS concepts
- Efficiency improvements for seL4 microkit and hardening of infrastructure code
- Build-out for assurance framework, traceability, attestation