Automated SysML v2 System Model to Memory-Safe Language Code Generation with Integrated AI Assistance

DARPA PROVERS

David S. Hardin, Ph.D.
Applied Research & Technology

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

Collaborators on this Presentation

- *Collins Aerospace*: Isaac Amundson, Junaid Babar, Darren Cofer, Saqib Hasan, Karl Hoech, Amer Tahat
- *Kansas State University*: Jason Belt, John Hatcliff, Robby
- *Aarhus University (Denmark)*: Stefan Hallerstede

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

DARPA PROVERS

Pipelined Reasoning of Verifiers Enabling Robust Systems

Develop automated, scalable **formal methods** tools that are integrated into traditional development pipelines using “proof engineering” techniques

Enable traditional product engineers to incrementally produce and maintain **high-assurance** national security systems

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

So How Did We Get Here?

DARPA High Assurance Cyber Military Systems (HACMS)

Loonwerks.com/projects/hacms

DEF CON 29

DARPA
@DARPA

We brought a hackable quadcopter with defenses built on our HACMS program to @defcon #AerospaceVillage. As program manager @raymondrichards reports, many attempts to breakthrough were made but none were successful. Formal methods FTW!

10:20 AM · Aug 9, 2021 · Hootsuite Inc.

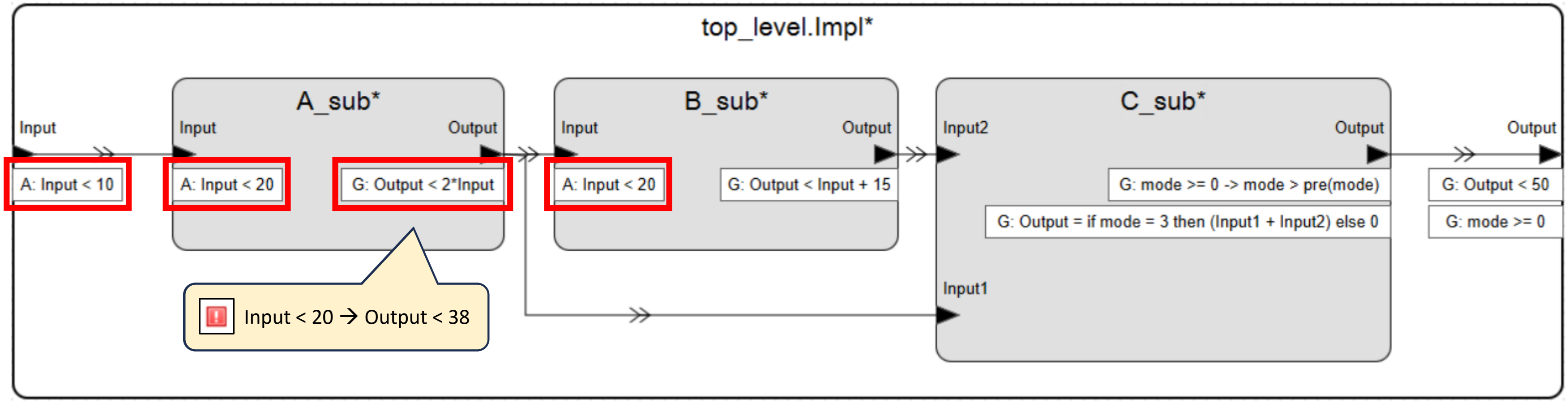
Collins Aerospace
An RTX Business

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

Compositional Reasoning for Model-Based Systems Engineering

Assume Guarantee REasoning Environment (AGREE)

- Assume-Guarantee annex for AADL architecture models
  - Assumptions describe the expectations that a component has on the environment
  - Guarantees describe bounds on the behavior of the component when assumptions are valid
- Compositional analysis to prove correctness of:
  - Component interfaces (component assumptions are satisfied by upstream guarantees)
  - Component implementations (component assumptions and subcomponent guarantees satisfy guarantees)



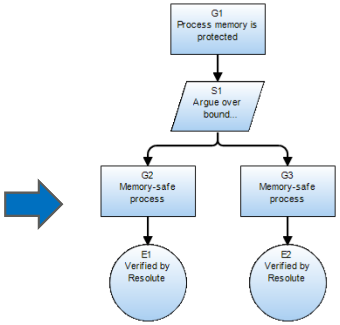
© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

Resolute

An Assurance Pattern Language and Evaluation Tool for Architecture Models

- The **structure of the system architecture** dictates the structure of the **assurance case**
  - Design patterns → Assurance patterns
- Extension of AADL language
  - Assurance case **instantiated** with elements from AADL model
- Specify **logical rules** for **evaluating evidence**
  - Automated evaluation

```
goal memory_protection(p : process) <=

  ** “Process " p " memory is protected from alterations by other processes" **

  strategy "Argue over bound processes";
  property(p, OS) = "seL4" or
   forall (mem : memory) . bound(p, mem) =>
     forall (q : process) . bound(q, mem) => memory_safe_process(q))
```

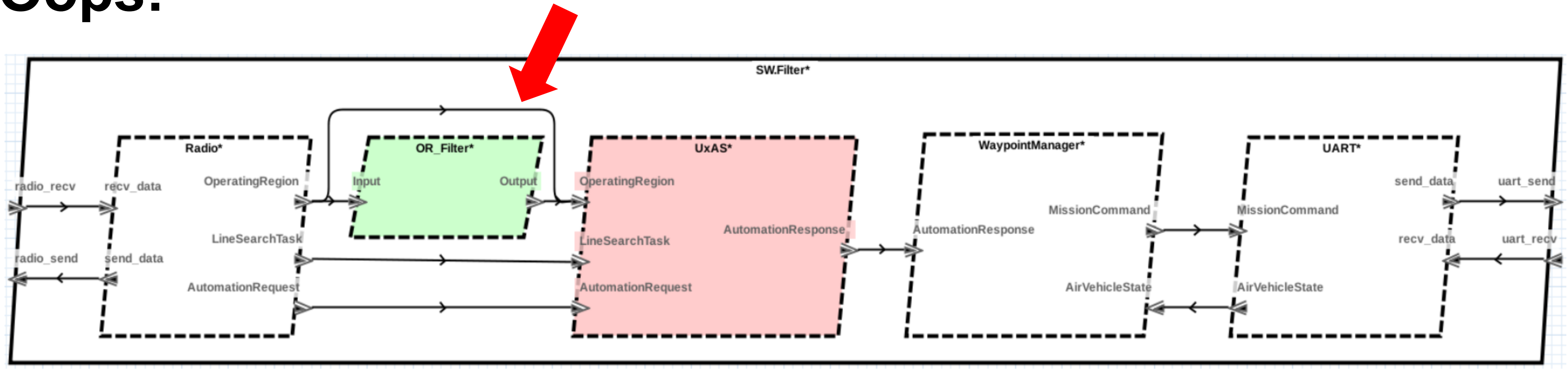


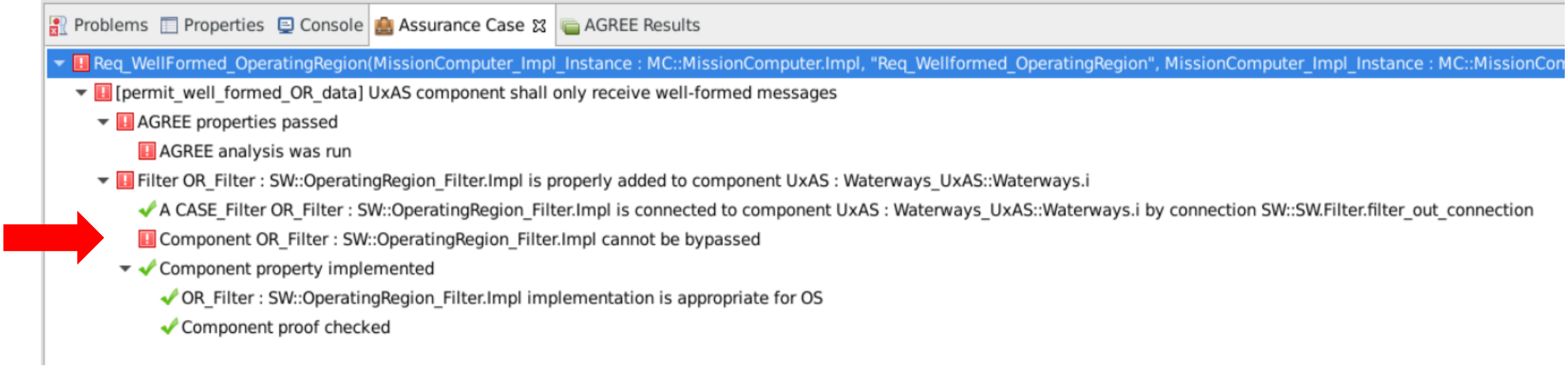
AdvoCATE (NASA)

Collins Aerospace
An RTX Business

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

Oops!





```
▾ ! Req_WellFormed_OperatingRegion(MissionComputer_Impl_Instance : MC::MissionComputer.Impl, "Req_Wellformed_OperatingRegion", MissionComputer_Impl_Instance : MC::MissionCon
  ▾ ! [permit_well_formed_OR_data] UxAS component shall only receive well-formed messages
    ▾ ! AGREE properties passed
        ! AGREE analysis was run
    ▾ ! Filter OR_Filter : SW::OperatingRegion_Filter.Impl is properly added to component UxAS : Waterways_UxAS::Waterways.i
        ✓ A CASE_Filter OR_Filter : SW::OperatingRegion_Filter.Impl is connected to component UxAS : Waterways_UxAS::Waterways.i by connection SW::SW.Filter.filter_out_connection
        ! Component OR_Filter : SW::OperatingRegion_Filter.Impl cannot be bypassed
      ▾ ✓ Component property implemented
          ✓ OR_Filter : SW::OperatingRegion_Filter.Impl implementation is appropriate for OS
          ✓ Component proof checked
```

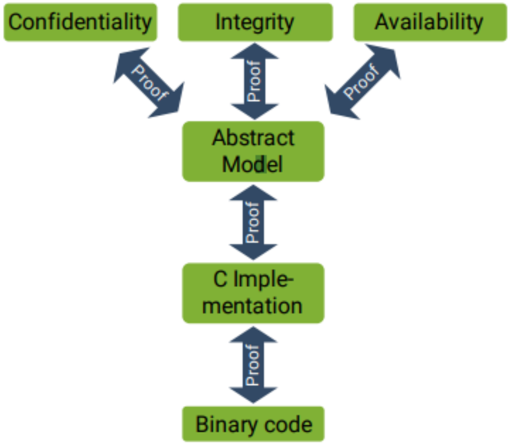
© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

seL4

Formally Verified Microkernel

University of New South Wales
Proofcraft

- seL4 microkernel guarantees partitioning of components and communication, backed by computer-checked proofs
- seL4 guarantees no infiltration, exfiltration, eavesdropping, interference, and provides fault containment for untrusted code
- Ensures soundness of the MBSE design process – components can be analyzed separately and composed safely



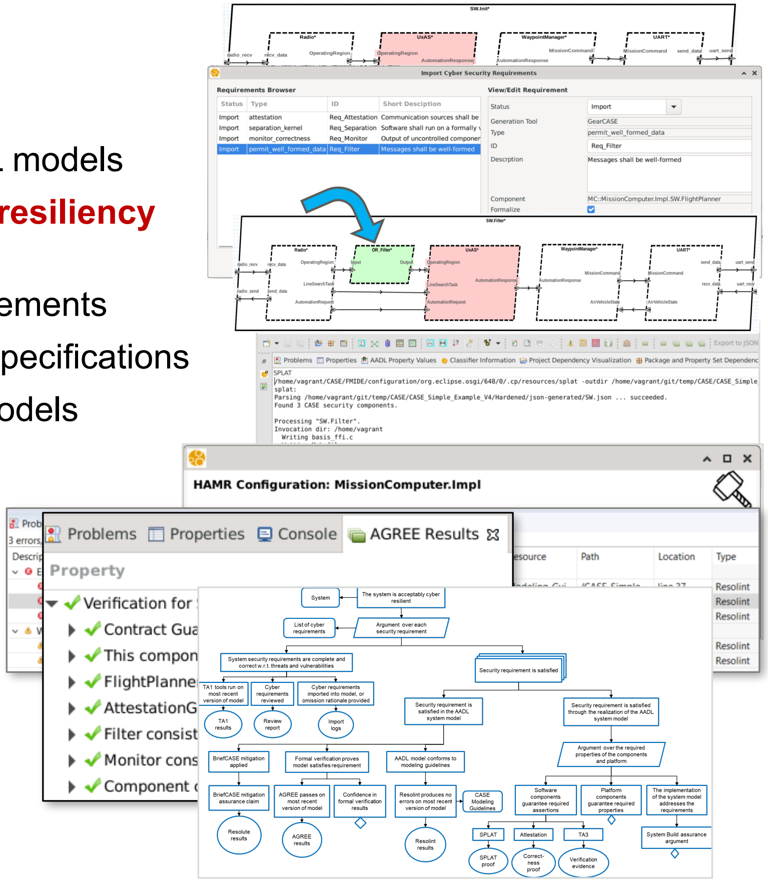
seL4 is…
- An operating system microkernel
- A hypervisor
- Proved correct
- Provably secure
- Fast

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

DARPA Cyber Assured Systems Engineering (CASE)

- Objective: develop the necessary design, analysis and verification tools that enable engineers to build cyber-resilient systems, including legacy elements

- **BriefCASE**
  - Integrated **model-based systems engineering** tool suite based on AADL models
  - Analyze architecture models for cyber vulnerabilities and generate **cyber resiliency requirements**
  - **Transform system architecture** models to satisfy cyber-resiliency requirements
  - Synthesize **high-assurance component implementations** from formal specifications
  - Generate **software integration code** directly from verified architecture models
  - Build to a **formally verified secure microkernel** target (seL4)
  - **Assurance**:
    - Check **model conformance** to standards
    - Verify system design and implementation using **formal methods**
    - Document proof of correctness with an **assurance case**


SW.Init*
Radio*
UxAS*
WaypointManager*
UART*
radio_recv
recv_data
OperatingRegion
AutomationResponse
MissionCommand
send_data
uart_send
Import Cyber Security Requirements
SW.Filter*
OR_Filter*
Input
Output
LineSearchTask
AutomationRequest
radio_send
recv_data
uart_recv
AirVehicleState
Problems
Properties
AADL Property Values
Classifier Information
Project Dependency Visualization
Package and Property Set Dependenc
Export to JSON
HAMR Configuration: MissionComputer.Impl
Console
AGREE Results
Property
Verification for
Contract Gua
This compon
FlightPlanne
AttestationG
Filter consist
Monitor cons
Component
3 errors
esource
Path
Location
Type
Resolint
Resolint
Resolint
Resolint
Resolint
System
The system is acceptably cyber resilient
List of cyber requirements
Argument over each security requirement
System security requirements are complete and correct w.r.t. threats and vulnerabilities
Security requirement is satisfied
TA1 tools run on most recent version of model
Cyber requirements reviewed
Cyber requirements imported into model, or omission rationale provided
TA1 results
Review report
Import logs
Security requirement is satisfied in the AADL system model
Security requirement is satisfied through the realization of the AADL system model
BriefCASE mitigation applied
Formal verification proves model satisfies requirement
AADL model conforms to modeling guidelines
Argument over the required properties of the components and platform
BriefCASE mitigation assurance claim
AGREE passes on most recent version of model
Confidence in formal verification results
Resolint produces no errors on most recent version of model
CASE Modeling Guidelines
Software components guarantee required assertions
Platform components guarantee required properties
The implementation of the system model addresses the requirements
Resolute results
AGREE results
Resolint results
SPLAT
Attestation
TA3
System Build assurance argument
SPLAT proof
Correct-ness proof
Verification evidence

Requirements Browser

Status	Type	ID	Short Desciption
Import	attestation	Req_Attestation	Communication sources shall be
Import	separation_kernel	Req_Separation	Software shall run on a formally v
Import	monitor_correctness	Req_Monitor	Output of uncontrolled componer
Import	permit_well_formed_data	Req_Filter	Messages shall be well-formed

View/Edit Requirement

Status	Import
Generation Tool	GearCASE
Type	permit_well_formed_data
ID	Req_Filter
Descrption	Messages shall be well-formed
Component	MC::MissionComputer.Impl.SW.FlightPlanner
Formalize	☑

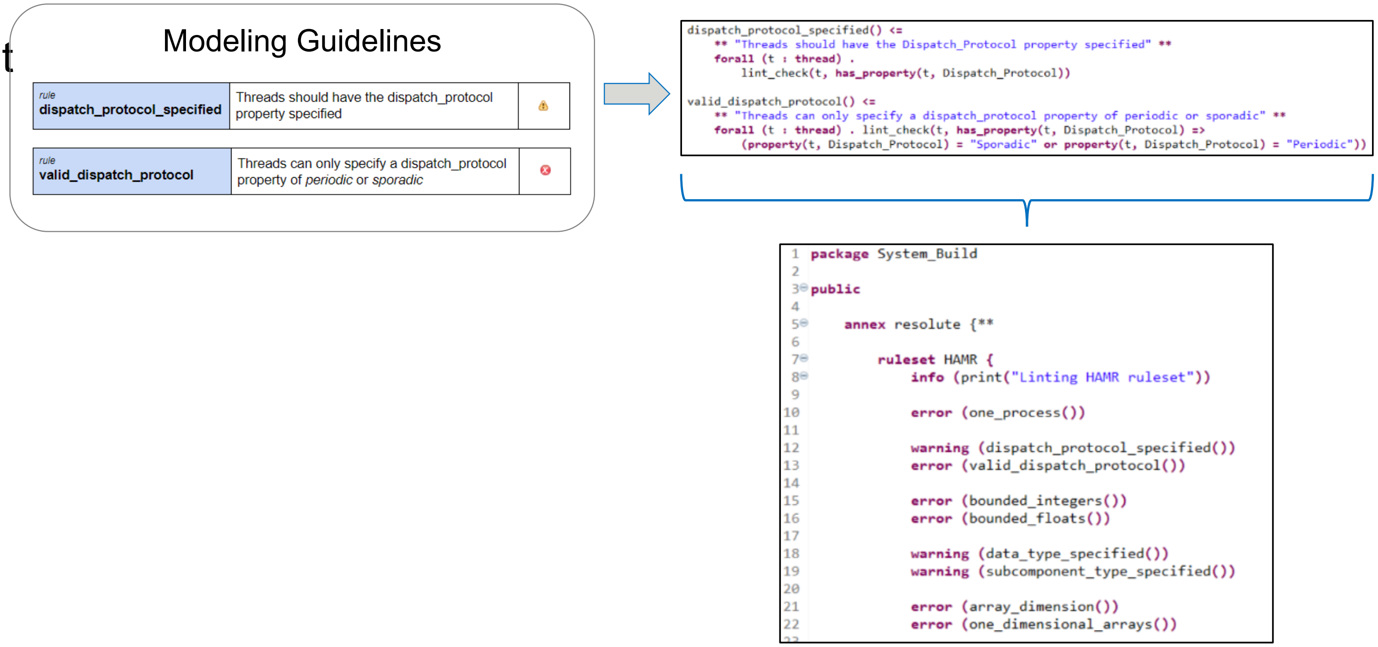
```
SPLAT
/home/vagrant/CASE/FMIDE/configuration/org.eclipse.osgi/648/0/.cp/resources/splat -outdir /home/vagrant/git/temp/CASE/CASE_Simple
splat:
Parsing /home/vagrant/git/temp/CASE/CASE_Simple_Example_V4/Hardened/json-generated/SW.json ... succeeded.
Found 3 CASE security components.

Processing "SW.Filter".
Invocation dir: /home/vagrant
  Writing basis_ffi.c
```

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

Resolint

A linter tool for AADL models

- Define *rules* in Resolute that correspond to modeling guidelines
- Group rules into *rulesets* corresponding to organizational process, customer requirements, certification guidelines, and tool constraints
- Automatically check compliance with modeling guidelines in OSATE

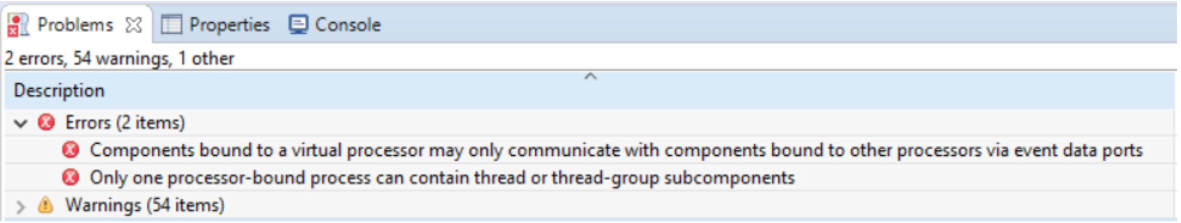


Modeling Guidelines

rule **dispatch_protocol_specified**	Threads should have the dispatch_protocol property specified	⚠
rule **valid_dispatch_protocol**	Threads can only specify a dispatch_protocol property of *periodic* or *sporadic*	⊗

```
dispatch_protocol_specified() <=
    ** "Threads should have the Dispatch_Protocol property specified" **
    forall (t : thread) .
        lint_check(t, has_property(t, Dispatch_Protocol))

valid_dispatch_protocol() <=
    ** "Threads can only specify a dispatch_protocol property of periodic or sporadic" **
    forall (t : thread) . lint_check(t, has_property(t, Dispatch_Protocol) =>
        (property(t, Dispatch_Protocol) = "Sporadic" or property(t, Dispatch_Protocol) = "Periodic"))
```

```
package System_Build

public

    annex resolute {**

        ruleset HAMR {
            info (print("Linting HAMR ruleset"))

            error (one_process())

            warning (dispatch_protocol_specified())
            error (valid_dispatch_protocol())

            error (bounded_integers())
            error (bounded_floats())

            warning (data_type_specified())
            warning (subcomponent_type_specified())

            error (array_dimension())
            error (one_dimensional_arrays())
```



```
2 errors, 54 warnings, 1 other
Description
Errors (2 items)
    Components bound to a virtual processor may only communicate with components bound to other processors via event data ports
    Only one processor-bound process can contain thread or thread-group subcomponents
Warnings (54 items)
```

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

Kansas State University

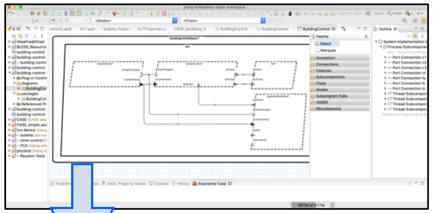
HAMR

High-Assurance Modeling and Rapid Engineering for Embedded Systems

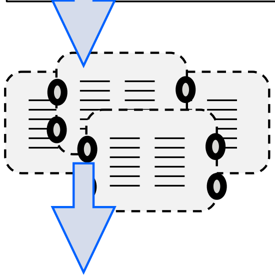
Modeling, analysis, and verification in AADL or SysML v2 modeling languages

Component development and verification in multiple languages

Deployments aligned with MBSE model semantics on multiple platforms



Leverage analyses from AADL community



- Slang (high integrity subset of Scala)
- C (utilizing memory-safe code generation)
- Rust (support on DARPA PROVERS)
- …

seL4 Deployment

Security. Performance. Proof.

verified microkernel

Linux Deployment

JVM Deployment

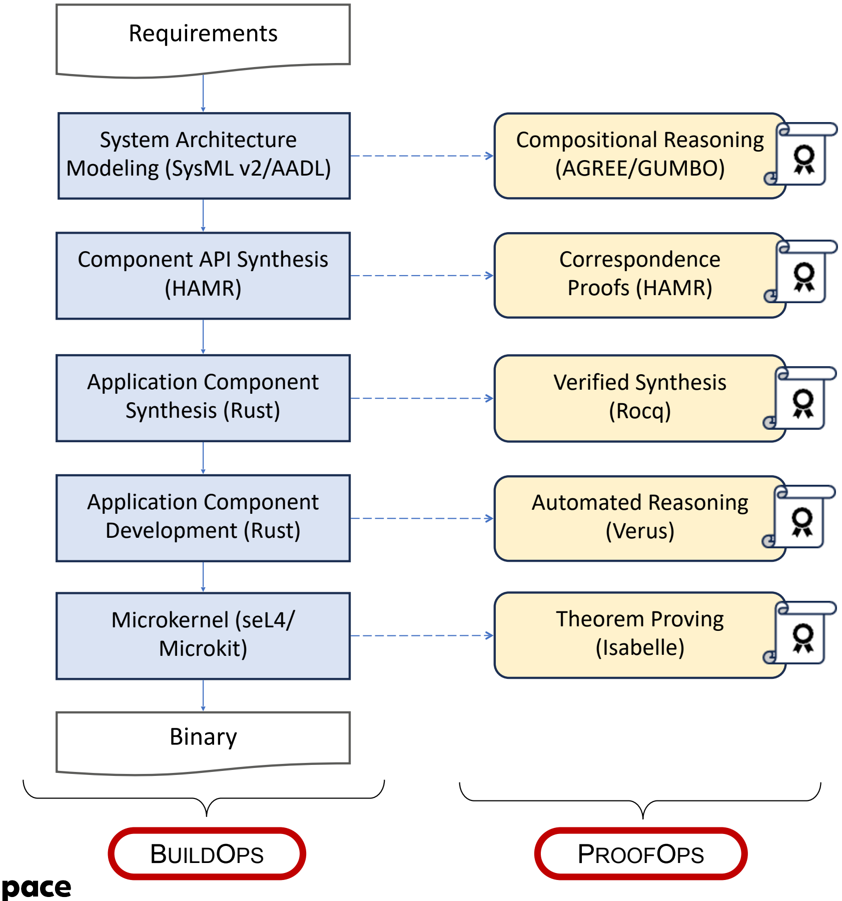
HAMR: infrastructure code generation and target platform build tool

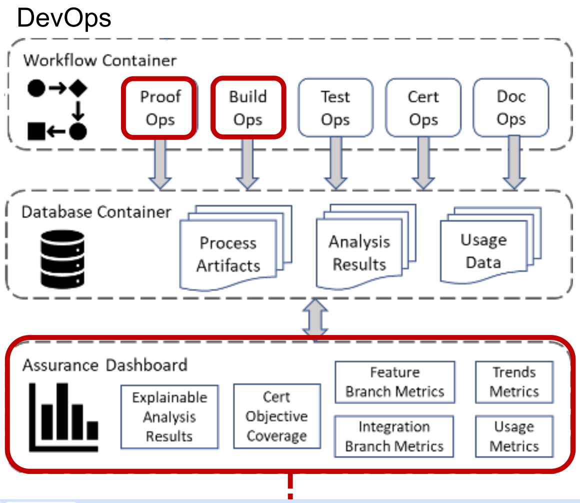
Tux logo: lewing@isc.tamu.edu

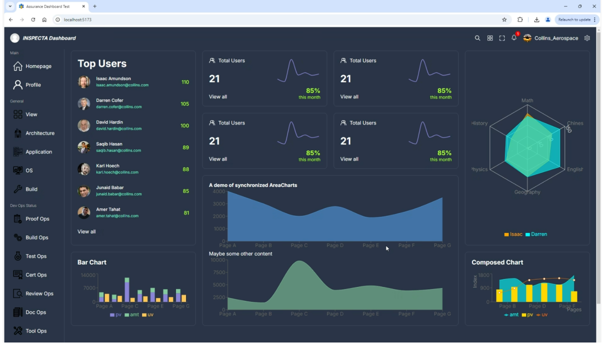
© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

DARPA PROVERS: INSPECTA Team

Industrial-Scale Proof Engineering for Critical Trustworthy Applications


Requirements
System Architecture Modeling (SysML v2/AADL)
Compositional Reasoning (AGREE/GUMBO)
Component API Synthesis (HAMR)
Correspondence Proofs (HAMR)
Application Component Synthesis (Rust)
Verified Synthesis (Rocq)
Application Component Development (Rust)
Automated Reasoning (Verus)
Microkernel (seL4/ Microkit)
Theorem Proving (Isabelle)
Binary
BUILDOPS
PROOFOPS





Top Users

Isaac Amundson	isaac.amundson@collins.com	110
Darren Cofer	darren.cofer@collins.com	105
David Hardin	david.hardin@collins.com	100
Saqib Hasan	saqib.hasan@collins.com	89
Karl Hoech	karl.hoech@collins.com	88
Junaid Babar	junaid.babar@collins.com	85
Amer Tahat	amer.tahat@collins.com	81

View all

Collins Aerospace
An RTX Business

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

INSPECTA Team

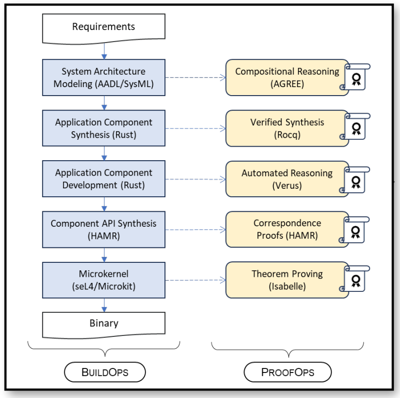
- Collins Aerospace, Team Lead
  - *Darren Cofer, PI*
- Carnegie Mellon University
- Dornerworks
- Kansas State University
  - with Aarhus University
- Proofcraft
- University of Kansas
- University of New South Wales

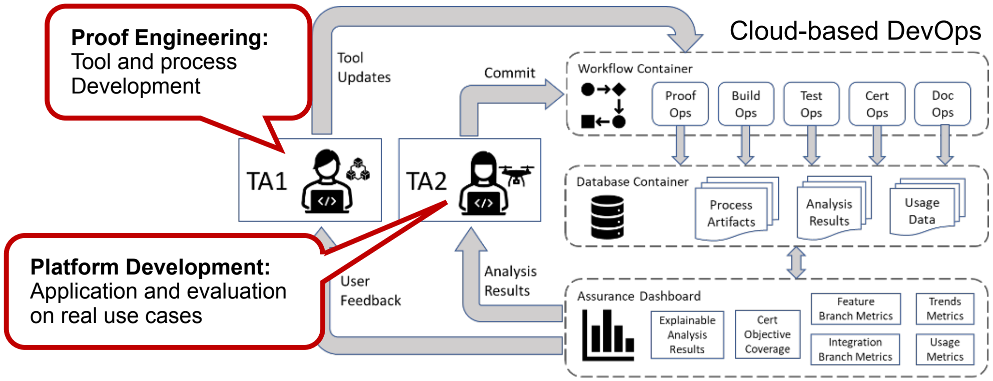
Collins Aerospace
An RTX Business

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

Technical Areas

TA1: Proof Engineering

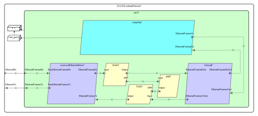

Requirements
System Architecture Modeling (AADL/SysML)
Application Component Synthesis (Rust)
Application Component Development (Rust)
Component API Synthesis (HAMR)
Microkernel (seL4/Microkit)
Binary
Compositional Reasoning (AGREE)
Verified Synthesis (Rocq)
Automated Reasoning (Verus)
Correspondence Proofs (HAMR)
Theorem Proving (Isabelle)
BuildOps
ProofOps


Proof Engineering: Tool and process Development

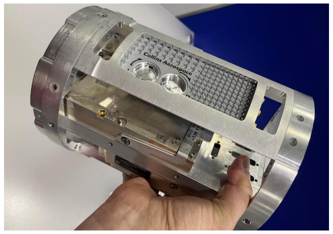
Platform Development: Application and evaluation on real use cases

TA2: Platform Development

- Open Platform
  - Developed and supported by DornerWorks
  - Unrestricted UAV mission software, system model with formal properties, multiple VMs, Rust software components, seL4 kernel



- Restricted Platform
  - Collins Launched Effects (LE) Mission Computer
  - Based on same computer hardware as the Open Platform

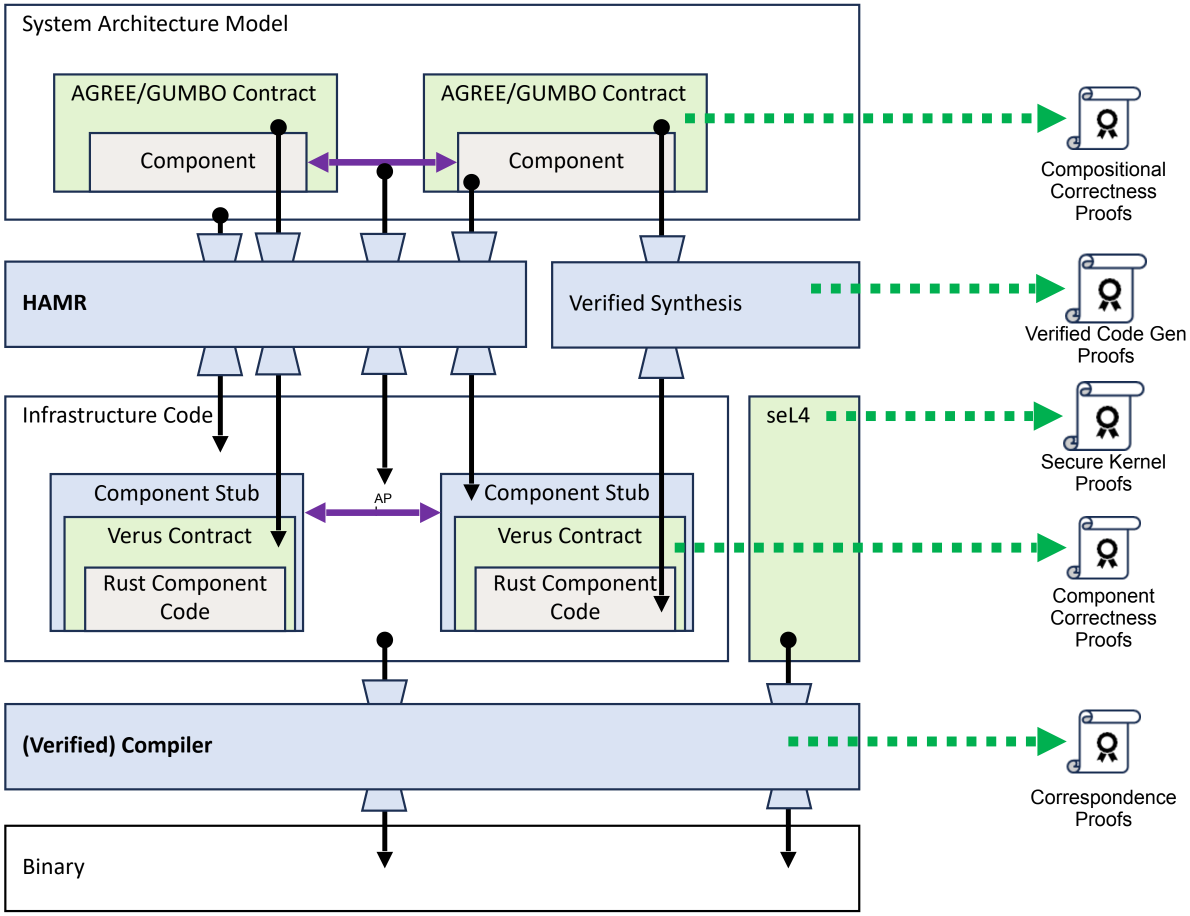


Collins Aerospace
An RTX Business

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

INSPECTA Proof Chain

With INSPECTA, engineers are able to generate comprehensive formal assurance across the entire development stack without requiring deep formal methods expertise



System Architecture Model

AGREE/GUMBO Contract

Component

AGREE/GUMBO Contract

Component

HAMR

Verified Synthesis

Infrastructure Code

Component Stub

Verus Contract

Rust Component Code

AP

Component Stub

Verus Contract

Rust Component Code

seL4

(Verified) Compiler

Binary

Compositional Correctness Proofs

Verified Code Gen Proofs

Secure Kernel Proofs

Component Correctness Proofs

Correspondence Proofs

Collins Aerospace
An RTX Business

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

SysML v2

- SysML v2 is the second major version of the Systems Modeling Language
  - Standarized under the auspices of the Object Management Group

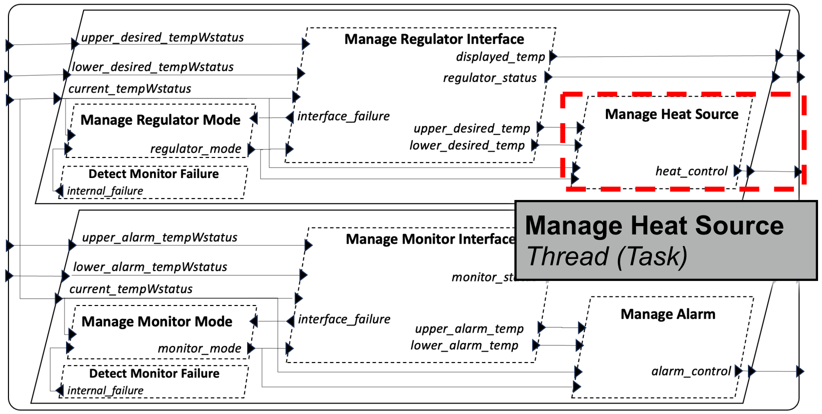
- Improved expressiveness relative to SysML v1
  - Now similar in expressiveness to AADL

- Standard textual form in addition to the graphical form
  - Promotes third party tool interaction

- Supported by major tool vendors: Siemens, The Mathworks, etc.
  - Necessary for mass adoption by the Defense Industrial Base

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

AADL to SysML v2 Transition Example



Manage Heat Source
Thread (Task)

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 + AADL Library

```
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;
```

Kansas State University

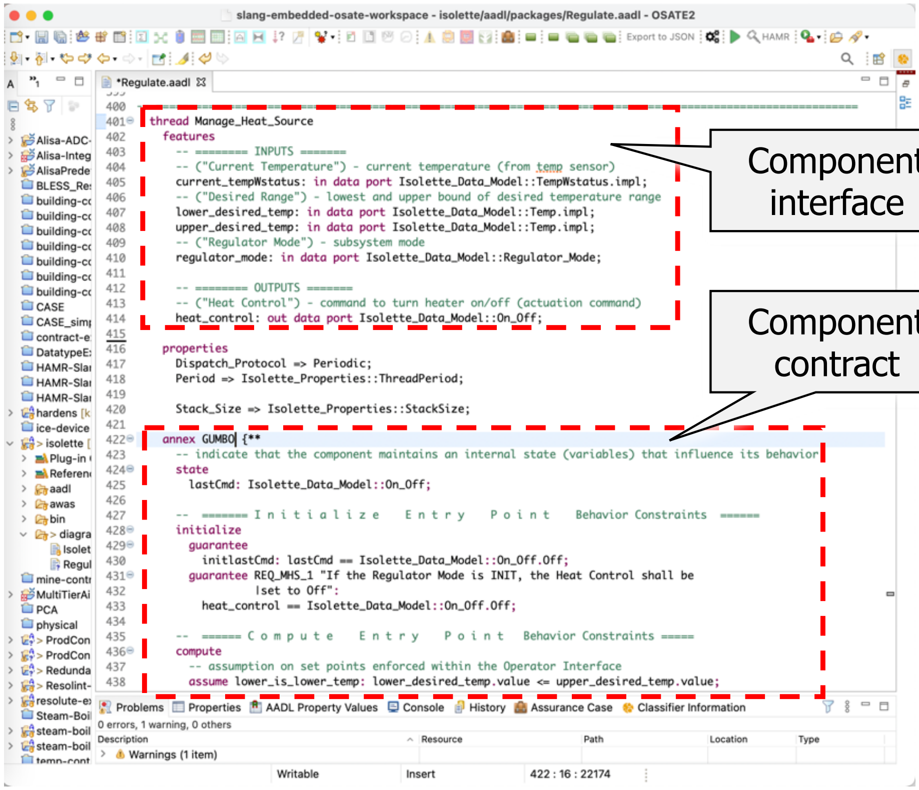
https://github.com/loonwerks/INSPECTA-models/tree/main/isolette/sysml

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

GUMBO Contract Language

- Inspired by AGREE and BLESS
- Aligns with MBSE run-time semantics
- Programming language independent
- Supports multiple quality assurance techniques
- Language Features:
  - Data type invariants
  - Port invariants (integration constraints)
  - Event-based / Shared-data based inter-thread communication
  - Local state declarations with invariants
  - Pre/Post conditions for thread code entry points
  - Support for fixed width scalars (e.g., Float32)

GUMBO contracts are specified in AADL/SysML v2 threads



```
--========================================================================================
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;

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

    Stack_Size => Isolette_Properties::StackSize;

  annex GUMBO {**
    -- indicate that the component maintains an internal state (variables) that influence its behavior
    state
      lastCmd: Isolette_Data_Model::On_Off;

    -- ======= I n i t i a l i z e   E n t r y   P o i n t   Behavior Constraints ======
    initialize
      guarantee
        initlastCmd: lastCmd == Isolette_Data_Model::On_Off.Off;
      guarantee REQ_MHS_1 "If the Regulator Mode is INIT, the Heat Control shall be
                 |set to Off":
        heat_control == Isolette_Data_Model::On_Off.Off;

    -- ====== C o m p u t e   E n t r y   P o i n t   Behavior Constraints =====
    compute
      -- assumption on set points enforced within the Operator Interface
      assume lower_is_lower_temp: lower_desired_temp.value <= upper_desired_temp.value;
```

Component interface

Component contract

Kansas State University / Galois

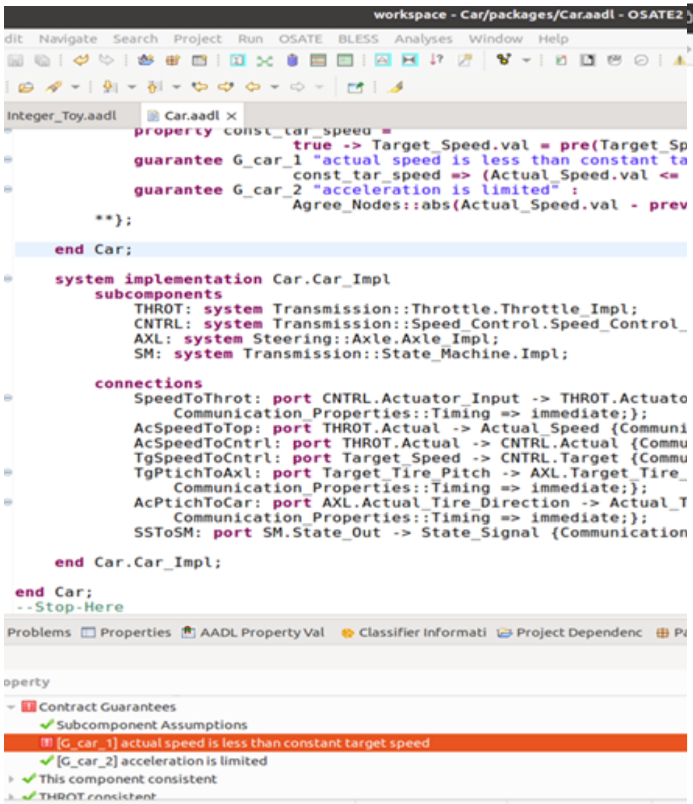
Collins Aerospace
An RTX Business

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

LLMs for MBSE Contract Verification: Counterexample Analysis/Resolution

- Counterexamples generated from MBSE contract verification can be difficult to analyze by non-experts
- We are utilizing LLMs to analyze these counterexamples, and suggest repairs
- Any LLM hallucinations are rejected, because assume/guarantee contract analysis is performed by a mathematically rigorous model checker
- We are also exploring use of LLMs for:
  - Proof repair
  - Documentation assistance
  - Model updates
  - Help writing formal properties

System Model

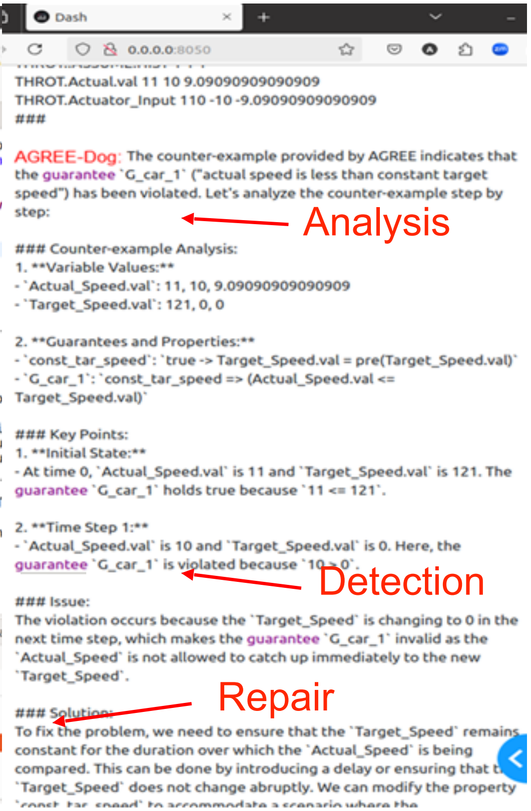

workspace - Car/packages/Car.aadl - OSATE2
dit Navigate Search Project Run OSATE BLESS Analyses Window Help
Integer_Toy.aadl
Car.aadl
Problems
Properties
AADL Property Val
Classifier Informati
Project Dependenc
operty

```
            property const_tar_speed =
                        true -> Target_Speed.val = pre(Target_Sp
            guarantee G_car_1 "actual speed is less than constant ta
                        const_tar_speed => (Actual_Speed.val <=
            guarantee G_car_2 "acceleration is limited" :
                        Agree_Nodes::abs(Actual_Speed.val - prev
        **};

    end Car;

    system implementation Car.Car_Impl
        subcomponents
            THROT: system Transmission::Throttle.Throttle_Impl;
            CNTRL: system Transmission::Speed_Control.Speed_Control_
            AXL: system Steering::Axle.Axle_Impl;
            SM: system Transmission::State_Machine.Impl;

        connections
            SpeedToThrot: port CNTRL.Actuator_Input -> THROT.Actuato
                Communication_Properties::Timing => immediate;};
            AcSpeedToTop: port THROT.Actual -> Actual_Speed {Communi
            AcSpeedToCntrl: port THROT.Actual -> CNTRL.Actual {Commu
            TgSpeedToCntrl: port Target_Speed -> CNTRL.Target {Commu
            TgPtichToAxl: port Target_Tire_Pitch -> AXL.Target_Tire_
                Communication_Properties::Timing => immediate;};
            AcPtichToCar: port AXL.Actual_Tire_Direction -> Actual_T
                Communication_Properties::Timing => immediate;};
            SSToSM: port SM.State_Out -> State_Signal {Communication

    end Car.Car_Impl;

end Car;
--Stop-Here
```

```
Contract Guarantees
    ✓ Subcomponent Assumptions
    [G_car_1] actual speed is less than constant target speed
    ✓ [G_car_2] acceleration is limited
✓ This component consistent
✓ THROT consistent
```

Contract counterexample

LLM prompts



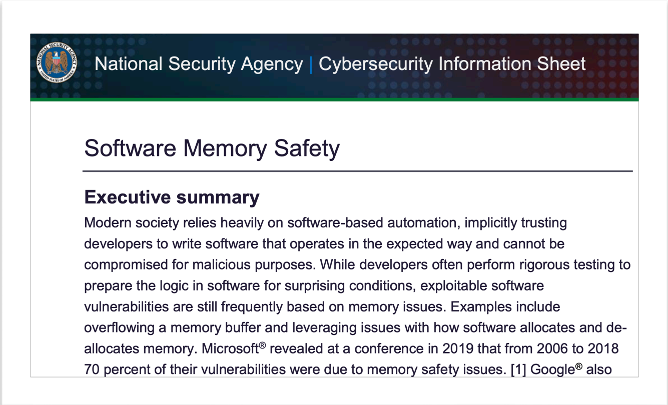
```
THROT.Actual.val 11 10 9.09090909090909
THROT.Actuator_Input 110 -10 -9.09090909090909
###

AGREE-Dog: The counter-example provided by AGREE indicates that
the guarantee `G_car_1` ("actual speed is less than constant target
speed") has been violated. Let's analyze the counter-example step by
step:

### Counter-example Analysis:
1. **Variable Values:**
- `Actual_Speed.val`: 11, 10, 9.09090909090909
- `Target_Speed.val`: 121, 0, 0

2. **Guarantees and Properties:**
- `const_tar_speed`: `true -> Target_Speed.val = pre(Target_Speed.val)`
- `G_car_1`: `const_tar_speed => (Actual_Speed.val <=
Target_Speed.val)`

### Key Points:
1. **Initial State:**
- At time 0, `Actual_Speed.val` is 11 and `Target_Speed.val` is 121. The
guarantee `G_car_1` holds true because `11 <= 121`.

2. **Time Step 1:**
- `Actual_Speed.val` is 10 and `Target_Speed.val` is 0. Here, the
guarantee `G_car_1` is violated because `10 > 0`.

### Issue:
The violation occurs because the `Target_Speed` is changing to 0 in the
next time step, which makes the guarantee `G_car_1` invalid as the
`Actual_Speed` is not allowed to catch up immediately to the new
`Target_Speed`.

### Solution:
To fix the problem, we need to ensure that the `Target_Speed` remains
constant for the duration over which the `Actual_Speed` is being
compared. This can be done by introducing a delay or ensuring that t
`Target_Speed` does not change abruptly. We can modify the property
`const_tar_speed` to accommodate a scenario where the
```

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

Memory-Safe Programming Languages

- An emerging consensus amongst computer science thought leaders is that memory-safe programming language technology needs to be adopted more broadly:

- “NSA recommends using a memory safe language when possible.” (Nov. 2022)
- The White House has published a report championing the adoption of memory safe programming languages to enhance software security. (Feb. 2024)
- Microsoft, Google, and Amazon have all announced significant Rust initiatives.
- Memory-safe language requirements are beginning to appear in U.S. Government contracting.

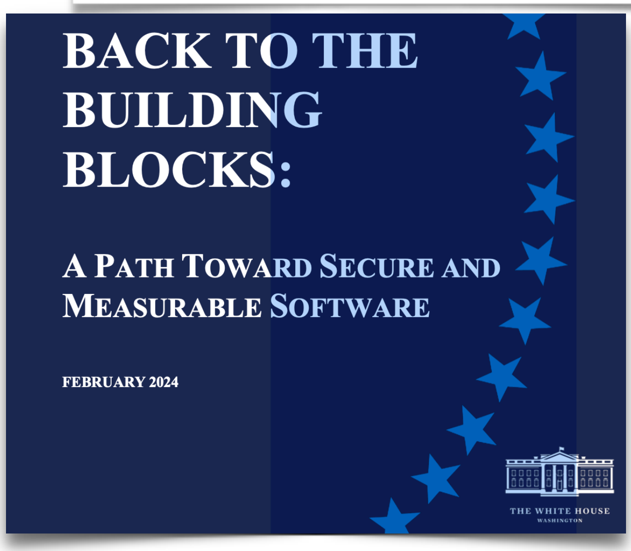


National Security Agency | Cybersecurity Information Sheet

Software Memory Safety

Executive summary

Modern society relies heavily on software-based automation, implicitly trusting developers to write software that operates in the expected way and cannot be compromised for malicious purposes. While developers often perform rigorous testing to prepare the logic in software for surprising conditions, exploitable software vulnerabilities are still frequently based on memory issues. Examples include overflowing a memory buffer and leveraging issues with how software allocates and de-allocates memory. Microsoft® revealed at a conference in 2019 that from 2006 to 2018 70 percent of their vulnerabilities were due to memory safety issues. [1] Google® also



BACK TO THE BUILDING BLOCKS:

A PATH TOWARD SECURE AND MEASURABLE SOFTWARE

FEBRUARY 2024

THE WHITE HOUSE
WASHINGTON

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

Why Memory-Safe Languages? Why now?

- Memory-safe languages are not new
  - For example, Collins successfully used Ada in major commercial and government avionics products in the 1980s and 1990s
  - Collins used SPARK effectively on high-assurance products for the intelligence community in the 2000s

- Recent improvements in compiler technology have made memory safety very low cost
- Additionally, novel memory ownership models (e.g, in Rust) have allowed references to be used safely
- Development organizations have tired of continual memory errors, causing a never-ending parade of security vulnerabilities, despite the use of increasingly sophisticated analysis tools

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

The Rust Programming Language

- The INSPECTA team is focusing our memory-safe language research on Rust

- Rust has several assurance advantages over C/C++, including:
  - Improved type safety
  - Vastly improved memory safety
  - No arbitrary pointer arithmetic
  - **…in short, 80% of C/C++ security flaws are eliminated outright!**

- Rust supports modern programming idioms such as a match primitive, traits, immutability by default, etc.

- Basic Rust syntax is familiar to C/C++ developers, easing the transition

- The Rust compiler produces code which is competitive in speed to C/C++

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

Verus: Rust Code Verification

- Verus is an open source Rust code verification environment under development by Carnegie Mellon University and numerous other researchers

- Verus has been utilized in a number of operating system, concurrent data structure, and distributed algorithm verification efforts

- Verus utilizes Rust syntax to express precondition and postcondition annotations, loop invariants, etc.

- Verus employs an SMT solver to attempt to prove postconditions, given the preconditions

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

Rust-Related Work on INSPECTA

- HAMR now supports the generation of Rust source code from SysML v2 models
  - For seL4, we use a new Rust userspace API developed by Nick Spinale
  - The KSU/Aarhus team is translating GUMBO system model contracts to the Verus Rust verification environment

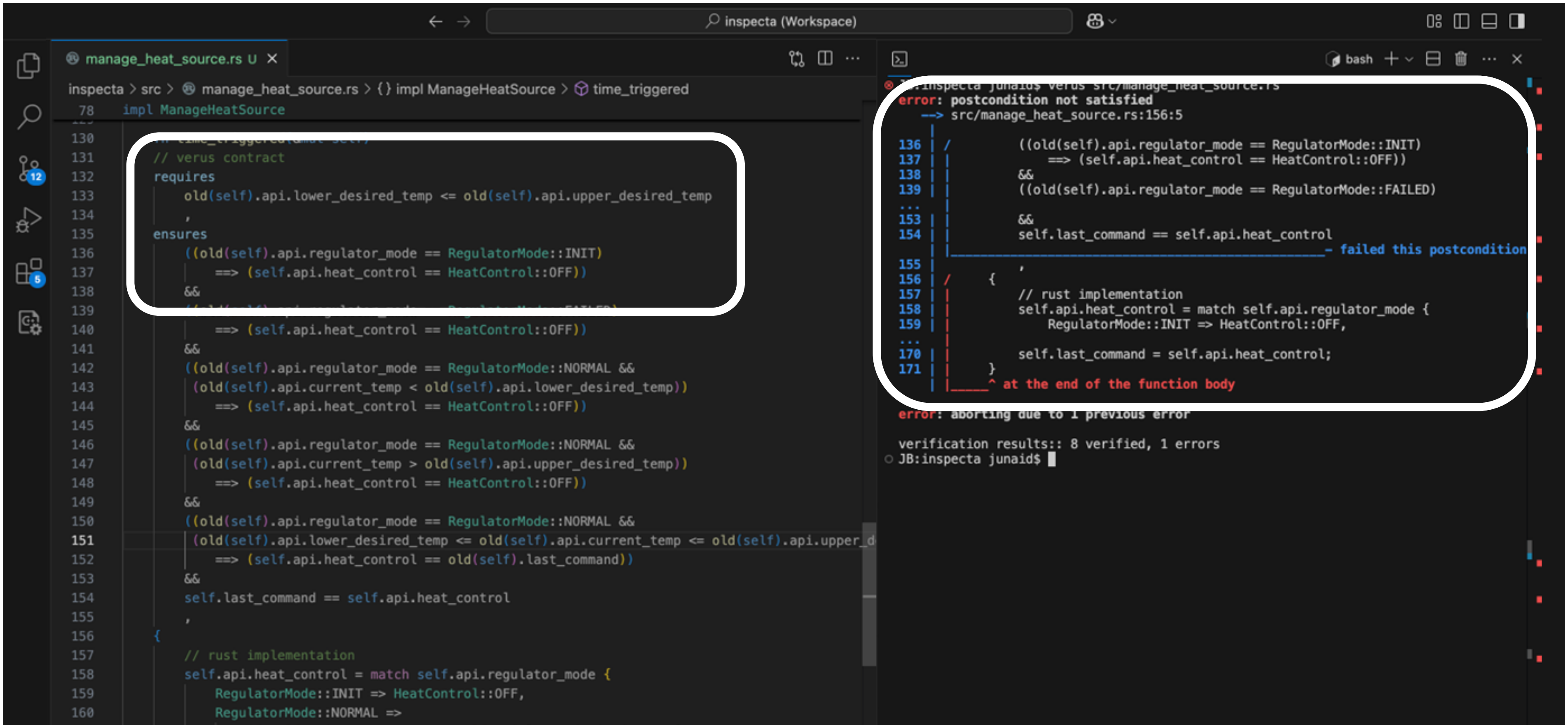
- The University of Kansas is developing Rust code generation for their attestation protocol specifications written in the Rocq theorem prover

- Dornerworks is writing open model application code in Rust

- CMU is enhancing Verus to support INSPECTA, reducing fragility in their SMT backend, and creating a connection to the Lean theorem prover

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

SysML v2 model with GUMBO contracts translated to Rust/Verus by the KSU HAMR tool


inspecta (Workspace)
manage_heat_source.rs U
inspecta > src > manage_heat_source.rs > {} impl ManageHeatSource > time_triggered
bash

```
impl ManageHeatSource
    // time_triggered(&mut self)
    // verus contract
    requires
        old(self).api.lower_desired_temp <= old(self).api.upper_desired_temp
        ,
    ensures
        ((old(self).api.regulator_mode == RegulatorMode::INIT)
            ==> (self.api.heat_control == HeatControl::OFF))
        &&
        ((old(self).api.regulator_mode == RegulatorMode::FAILED)
            ==> (self.api.heat_control == HeatControl::OFF))
        &&
        ((old(self).api.regulator_mode == RegulatorMode::NORMAL &&
         (old(self).api.current_temp < old(self).api.lower_desired_temp))
            ==> (self.api.heat_control == HeatControl::OFF))
        &&
        ((old(self).api.regulator_mode == RegulatorMode::NORMAL &&
         (old(self).api.current_temp > old(self).api.upper_desired_temp))
            ==> (self.api.heat_control == HeatControl::OFF))
        &&
        ((old(self).api.regulator_mode == RegulatorMode::NORMAL &&
         (old(self).api.lower_desired_temp <= old(self).api.current_temp <= old(self).api.upper_d
            ==> (self.api.heat_control == old(self).last_command))
        &&
        self.last_command == self.api.heat_control
        ,
    {
        // rust implementation
        self.api.heat_control = match self.api.regulator_mode {
            RegulatorMode::INIT => HeatControl::OFF,
            RegulatorMode::NORMAL =>
```

```
JB:inspecta junaid$ verus src/manage_heat_source.rs
error: postcondition not satisfied
   --> src/manage_heat_source.rs:156:5
    |
136 | /         ((old(self).api.regulator_mode == RegulatorMode::INIT)
137 | |             ==> (self.api.heat_control == HeatControl::OFF))
138 | |         &&
139 | |         ((old(self).api.regulator_mode == RegulatorMode::FAILED)
...   |
153 | |         &&
154 | |         self.last_command == self.api.heat_control
    | |__________________________________________________- failed this postcondition
155 |           ,
156 | /     {
157 | |         // rust implementation
158 | |         self.api.heat_control = match self.api.regulator_mode {
159 | |             RegulatorMode::INIT => HeatControl::OFF,
...   |
170 | |         self.last_command = self.api.heat_control;
171 | |     }
    | |_____^ at the end of the function body

error: aborting due to 1 previous error

verification results:: 8 verified, 1 errors
JB:inspecta junaid$
```

Collins Aerospace
An RTX Business

Carnegie Mellon University

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

Conclusion

- The INSPECTA team is **making formal verification** across the entire software development stack **accessible to non-formal methods experts** through automated analysis, DevOps integration, a ProofOps console, and improved user feedback

- Keys to achieving this goal include integration with the SysML v2 System Modeling Language, and support for modern memory-safe languages, specifically Rust

- Much important INSPECTA Research was not mentioned in this talk, including:
  - AGREE/GUMBO contract language harmonization (Collins / KSU)
  - Verified Component Synthesis (KU)
  - Lifecycle Attestation (KU / Collins)
  - seL4 proof engineering (Proofcraft), Microkit, and Lions OS (UNSW)

- Check it out – code, papers, links:
  - **https://loonwerks.com/projects/inspecta.html**

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.

Thank You!

Contact: david.hardin@collins.com

This work was funded by DARPA contract FA8750-24-9-1000.

The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

© 2025 Collins Aerospace. | This document does not include any export controlled technical data.