High Assurance Modeling and Rapid Engineering (HAMR) for Embedded Systems

AADL Tool Expo – October 28, 2019

John Hatcliff

University Distinguished Professor
Lucas-Rathbone Professor of Engineering
Kansas State University

Robby

Professor
Kansas State University

Jason Belt,
Hariharan Thiagarajan

Research Associates
Kansas State University

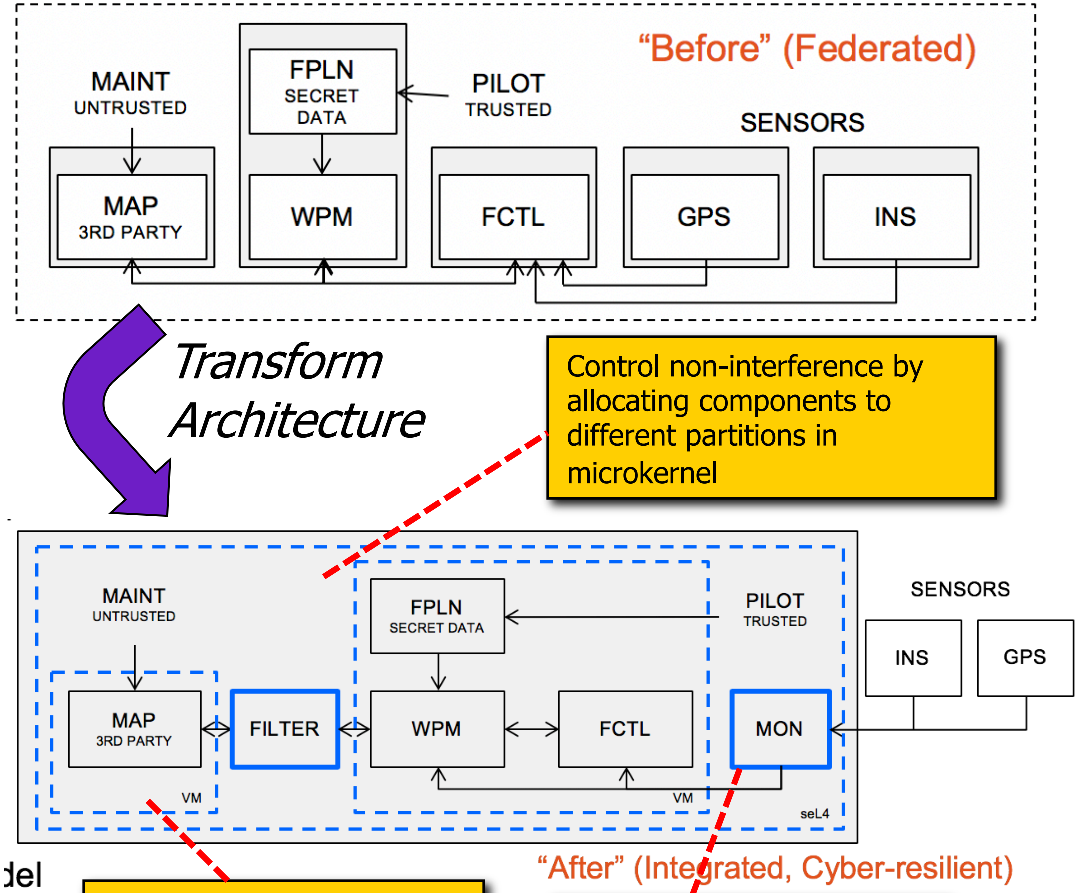
In collaboration with Adventium Labs, SEI, and Collins Aerospace

This material is based on research sponsored by the Department of Homeland Security (DHS) Science and Technology Directorate, Homeland Security Advanced Research Projects Agency (HSARPA), Cyber Security Division (DHS S&T/HSARPA/CDS) BAA HSHQDC- 14-R-B0005, the Government of Israel and the National Cyber Bu- reau in the Government of Israel via contract number D16PC00057, as well as the US National Science Foundation FDA Scholar-in-Residence Program.

DARPA CASE Approach

DARPA CASE provides tools to develop **cyber-resiliency requirements**, **refactor/transform system architectures**, and **generate code/builds** of modified systems that achieve cyber-resiliency

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



Control non-interference by allocating components to different partitions in microkernel

Wrap legacy untrusted component in a VM in micro-kernel partition

Insert attestation managers to ensure data is coming from a trusted source.

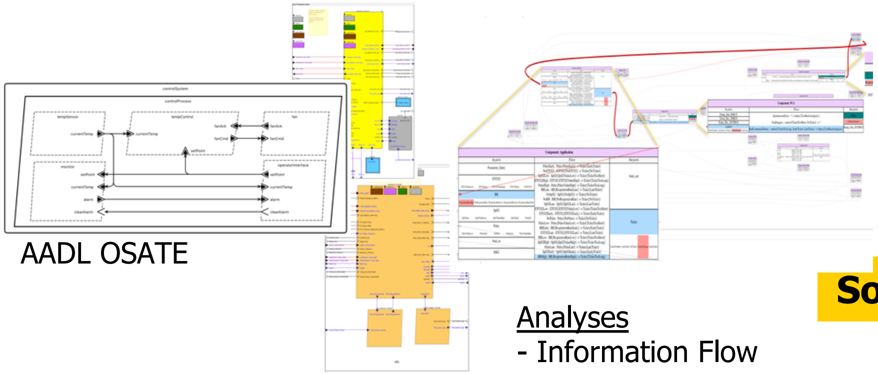
On DARPA CASE, KSU is partnered with Adventium Labs, Collins Aerospace, Data61 (SeL4 verified microkernel)

Deeply Integrate Models and Programming Across Multiple Levels of Abstraction

Analysis and verification results moved up and down abstraction layers

Semantic Consistency

System **Modeling and Analysis** (AADL)

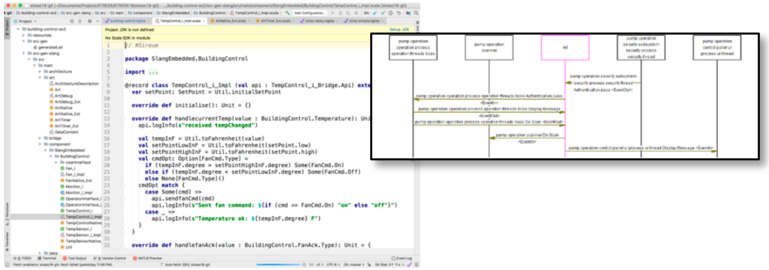


AADL OSATE

Code Generation
-- Slang + AADL Run-Time Reference Implementation

Analyses
- Information Flow
- Functional Integration Constraints (component contracts)
- Scheduleability
- …

Source Code, Simulation, Analysis, Verification



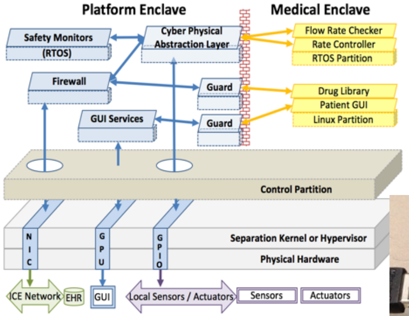
Project JDK is not defined

No Scala SDK in module

```
// #Sireum

package SlangEmbedded.BuildingControl

import ...

@record class TempControl_i_Impl (val api : TempControl_i_Bridge.Api) exte
  var setPoint: SetPoint = Util.initialSetPoint

  override def initialise(): Unit = {}

  override def handlecurrentTemp(value : BuildingControl.Temperature): Uni
    api.logInfo(s"received tempChanged")

    val tempInF = Util.toFahrenheit(value)
    val setPointLowInF = Util.toFahrenheit(setPoint.low)
    val setPointHighInF = Util.toFahrenheit(setPoint.high)
    val cmdOpt: Option[FanCmd.Type] =
      if (tempInF.degree > setPointHighInF.degree) Some(FanCmd.On)
      else if (tempInF.degree < setPointLowInF.degree) Some(FanCmd.Off)
      else None[FanCmd.Type]()
    cmdOpt match {
      case Some(cmd) =>
        api.sendfanCmd(cmd)
        api.logInfo(s"Sent fan command: ${if (cmd == FanCmd.On) "on" else "off"}")
      case _ =>
        api.logInfo(s"Temperature ok: ${tempInF.degree} F")
    }
  }

  override def handlefanAck(value : BuildingControl.FanAck.Type): Unit = {
```

Slang – Subset of Scala for critical systems

Deployment on Embedded/Distributed Platforms



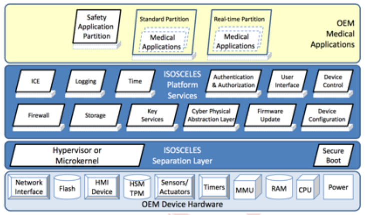
Partitioned Architectures

Micro-kernels & OS
- SeL4
- Minex 3 (enhanced)
- Xen
- Linux
- FreeRTOS

Code Generation, e.g.,
- C + Platform Run-Time System (primitives for controlling communication between partitions in a partitioning architecture)
- C compatible with CompCERT verified compiler

Example Domains

Medical Devices *(US Dept of Homeland Security)*



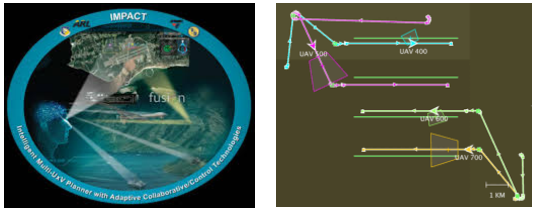
Code deployed using Genode OS framework using Xen Hypervisor and SeL4 microkernel

Building Controls (US Dept of Homeland Security)

Containment labs for critical agriculture experiments

Code deployed using enhanced Minix 3 micro-kernel

UxAS – Unmanned *(AFRL, DARPA)*


IMPACT
fusion
Intelligent Multi-UxV Planner with Adaptive Collaborative/Control Technologies
UAV 500
UAV 400
UAV 600
UAV 700
1 KM

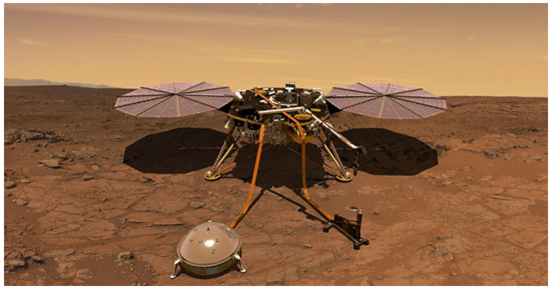
Unmanned Systems Autonomy Services

Code deployed on machine-verified micro-kernel SEL4

seL4
Security. Performance. Proof.

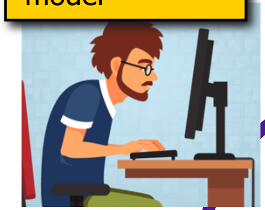
- Targetting development and verification of embedded systems
- Emphasizing platform development on using separation kernel and hypervisor technology
- Introduce rigorous use of modeling and abstractions without significant disruption of workflows

NASA/JPL



AADL Computational Model

Developer configures computational model

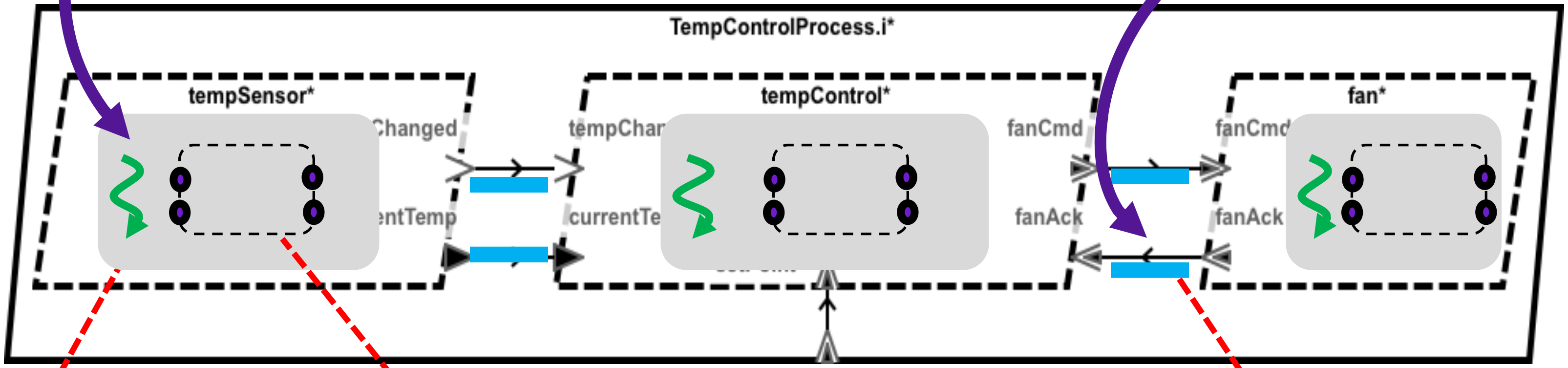


AADL Thread Property Options

Periodic
Sporadic
Hybrid
…

AADL Port & Connection Property Options

Event
Data
Temporal
  Separation
…

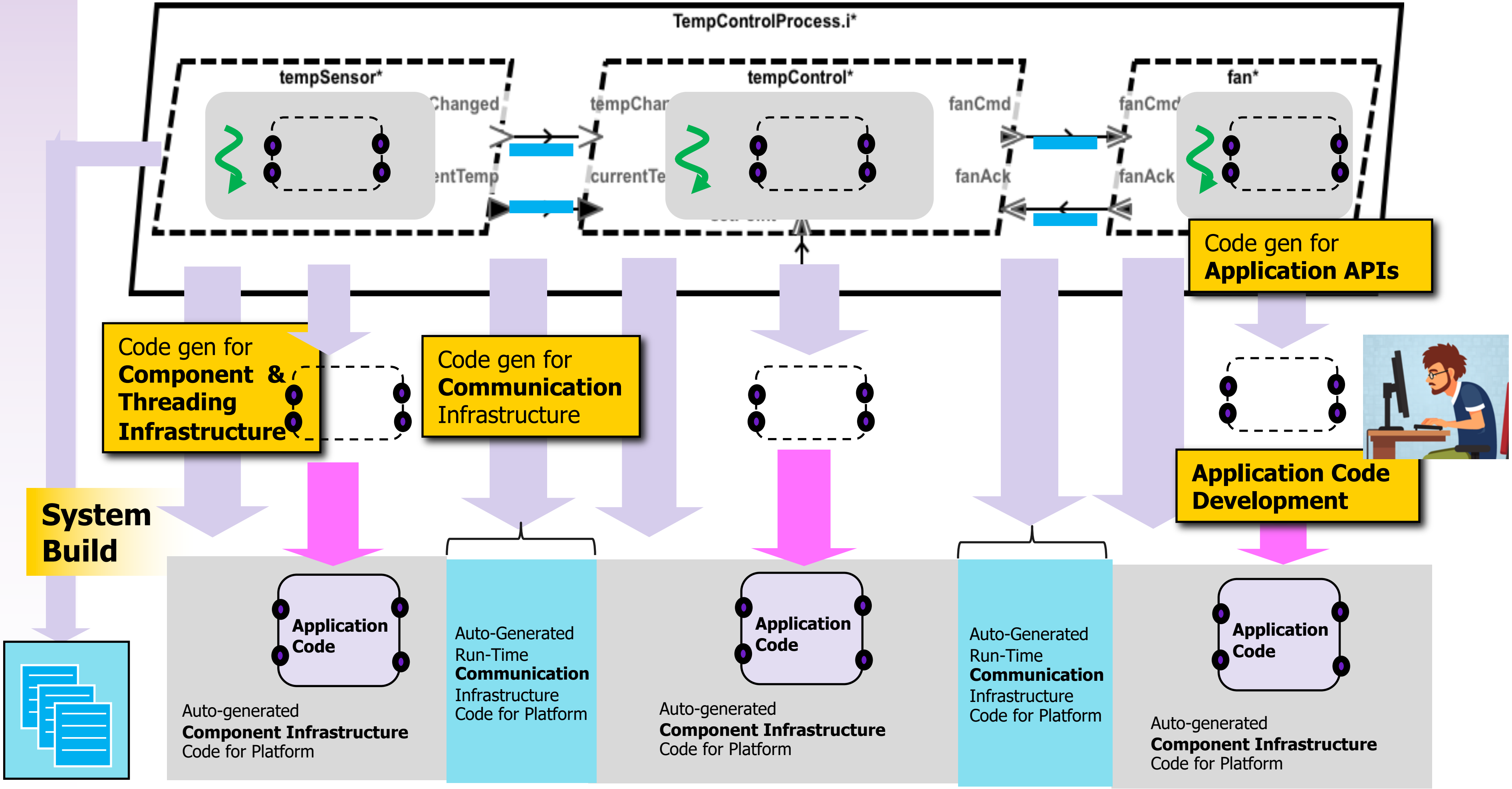


Selected thread pattern

Implied API Pattern for application code to access AADL Run-Time Services

Selected communication pattern

HAMR Code Generation



Code gen for **Application APIs**

Code gen for **Component & Threading Infrastructure**

Code gen for **Communication** Infrastructure

Application Code Development

System Build

Auto-generated **Component Infrastructure** Code for Platform

Auto-Generated Run-Time **Communication** Infrastructure Code for Platform

Auto-generated **Component Infrastructure** Code for Platform

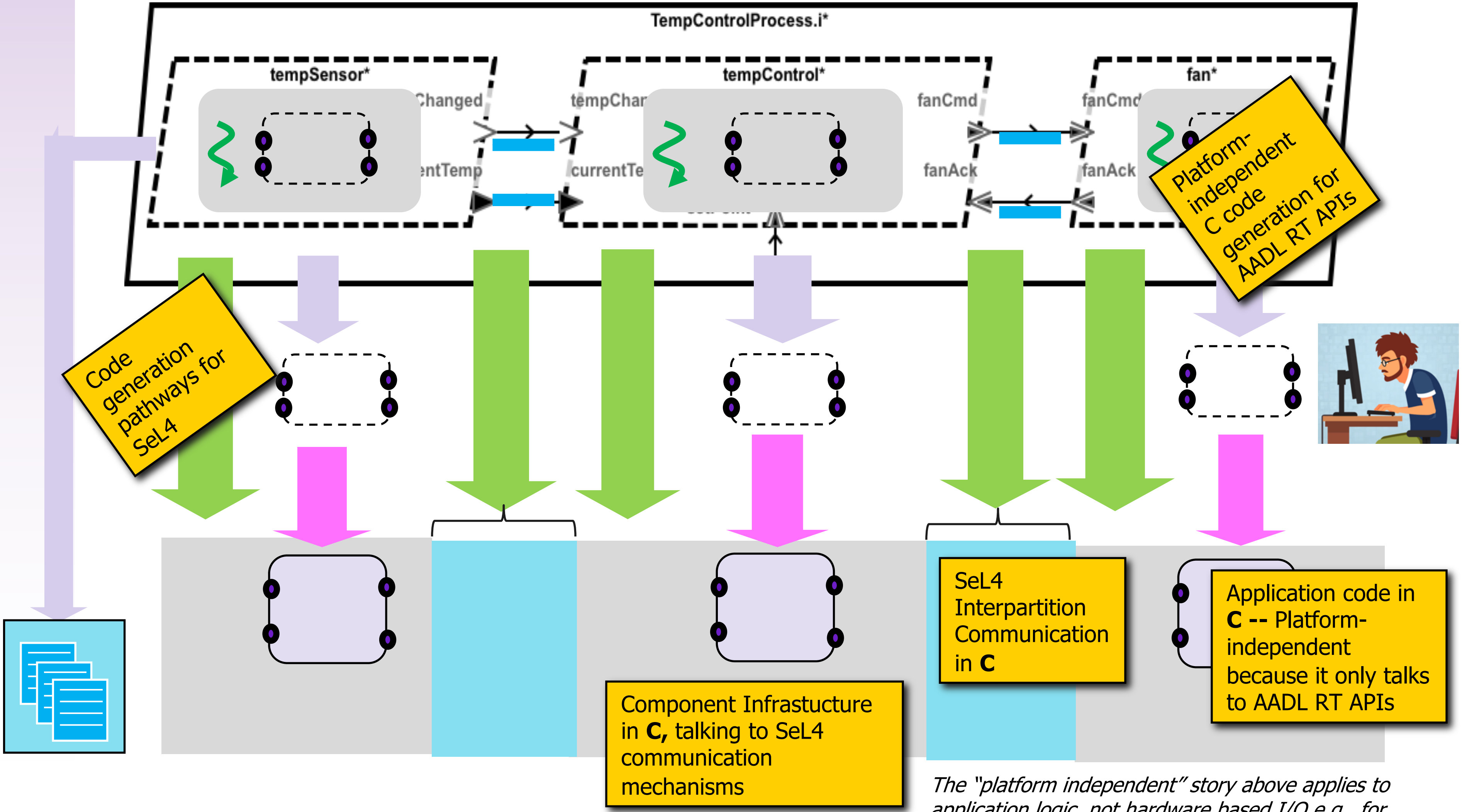
Auto-Generated Run-Time **Communication** Infrastructure Code for Platform

Auto-generated **Component Infrastructure** Code for Platform

Platform configuration information

HAMR Code Generation

Use Case: Example HAMR instantiation for C-based development on **SeL4 microkernel** (e.g., DARPA CASE)



Platform-independent C code generation for AADL RT APIs

Code generation pathways for SeL4

SeL4 Interpartition Communication in **C**

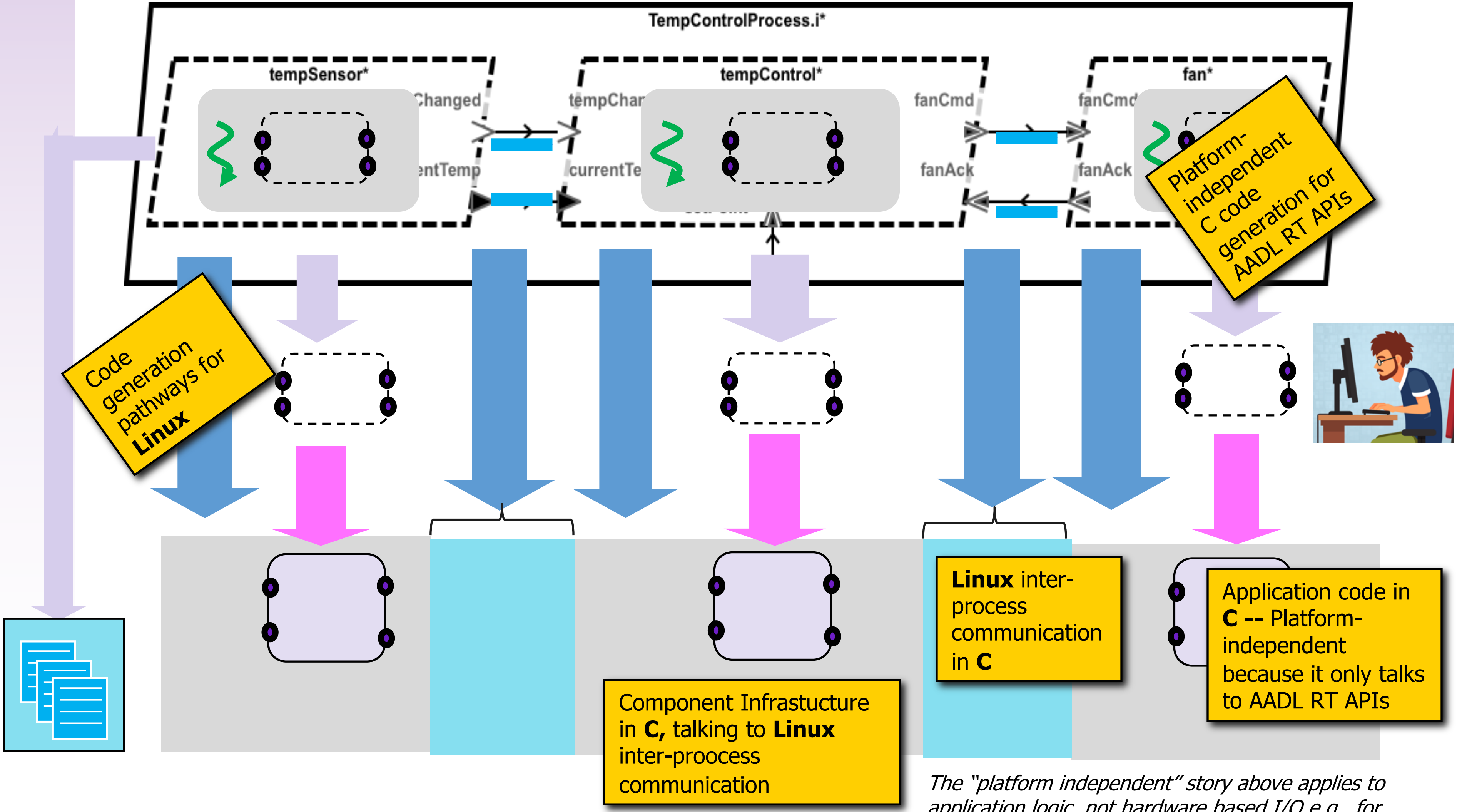
Application code in **C --** Platform-independent because it only talks to AADL RT APIs

Component Infrastucture in **C,** talking to SeL4 communication mechanisms

The "platform independent" story above applies to application logic, not hardware based I/O e.g., for sensors, actuators.

HAMR Code Generation

Use Case: Example HAMR instantiation for C-based development on **Linux** (e.g., DARPA CASE)



Platform-independent C code generation for AADL RT APIs

Code generation pathways for **Linux**

Linux inter-process communication in **C**

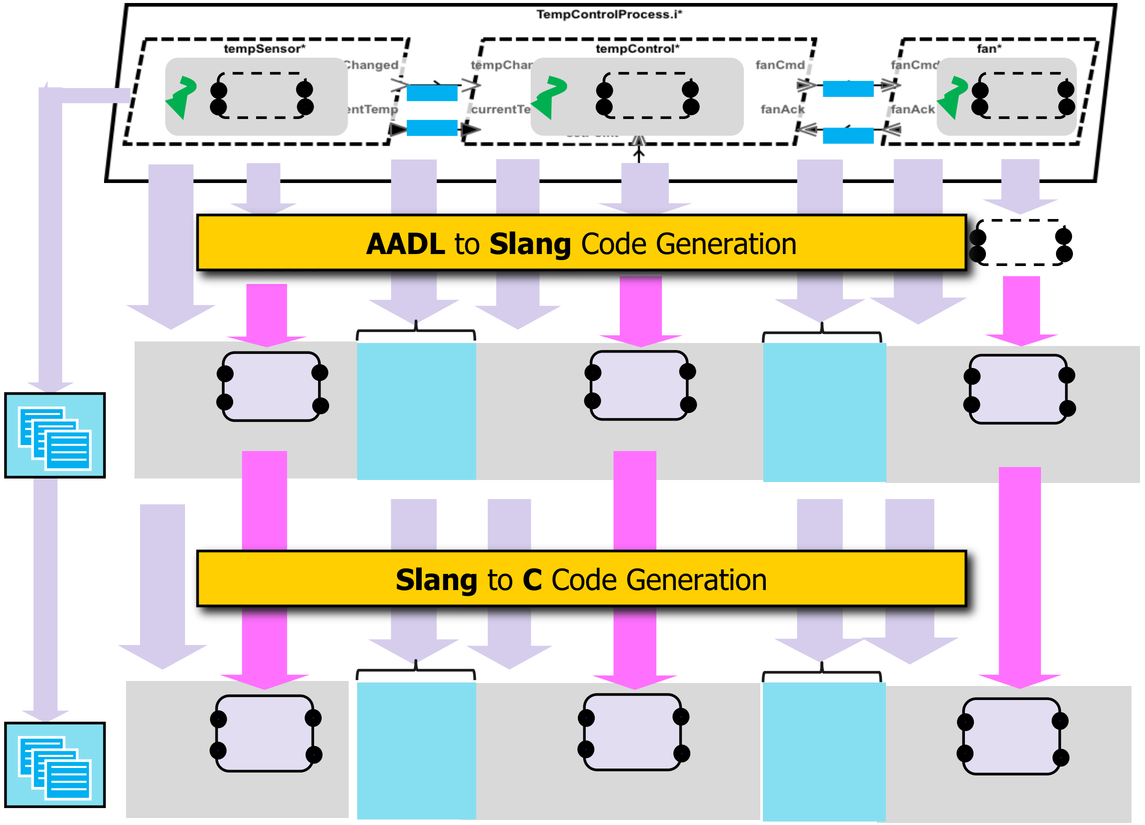
Application code in **C --** Platform-independent because it only talks to AADL RT APIs

Component Infrastucture in **C,** talking to **Linux** inter-proocess communication

The "platform independent" story above applies to application logic, not hardware based I/O e.g., for sensors, actuators.

HAMR Code Generation

Use Case: High-Assurance Development in **Slang**, with a C-based deployment



AADL to **Slang** Code Generation

Slang to **C** Code Generation

System **Modeling and Analysis**

...in AADL

Source Code, Simulation, Analysis, Verification

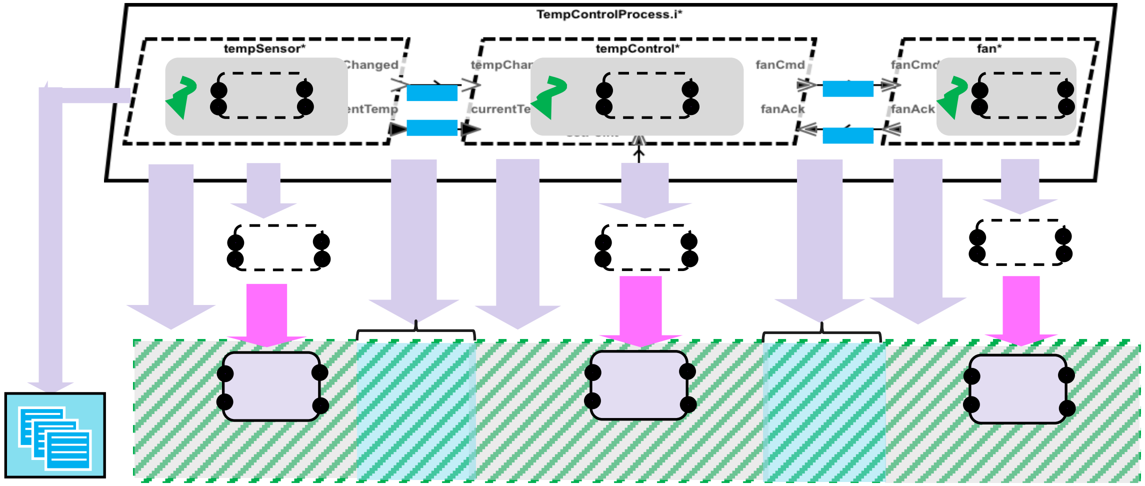
*...in **Slang** – a safety-critical subset of Scala*

Deployment on Embedded/Distributed Platforms

*...ie.g., in **C** with platform infrastructure*

HAMR Run-time Reference Implementation

The Slang-based infrastructure of AADL run-time provides a reference implementation



System Modeling and Analysis

...in AADL

Reference Implementation for AADL Computational Model in Slang

- HAMR AADL reference implementation is analogous to an **abstract machine for analyzeable real-time embedded computation**
- Because Slang (subset of Scala) is a JVM-based language it is easy to integrate with Java resources to obtain a **simulation, visualization, and run-time verification environment** for AADL-derived applications
    - Sensor, actuator, UI elements not a part of core application logic can be mocked up in Java or Scala

High Assurance High-Level Development in Slang (subset of Scala)

In addition to supporting C development, we also support "higher-level" development in Slang (subset of Scala) which supports integration with Java

- Slang -- A verifiable subset of a modern programming language — Scala
  - **imperative OO & FP**: generics, pattern matching, higher-order functions, etc.
  - **benefits:** existing Java ecosystems and talent pools, available (customizable) industrial tool support, including compiler toolchain & IDEs
  - … yet able to generate code suitable for safety/security-critical embedded systems
- (Currently) supports two **memory models**:
  - SPARK/Ada-like (static memory allocation): targeted for embedded systems
  - Swift-like (DAG, immutable sharing, automatic reference counting): targeted for large-application development
    - including for developing Sireum/Slang itself!

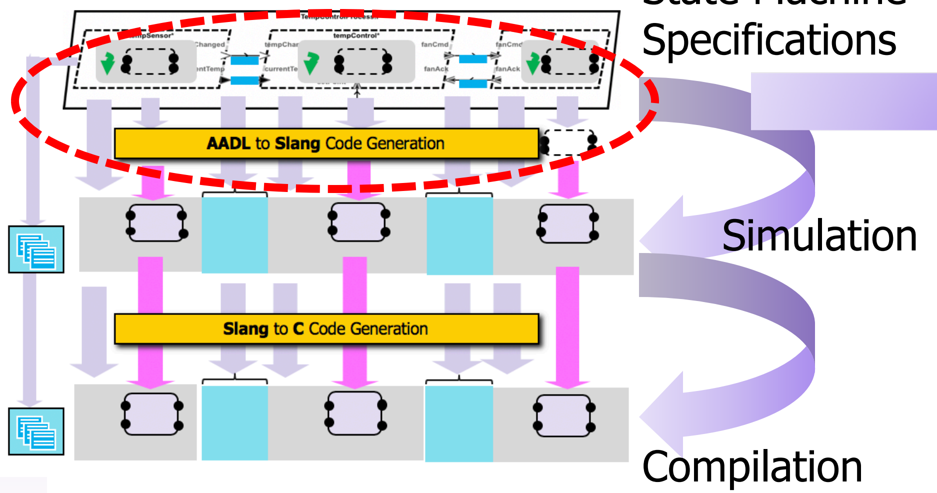
Slang-to-C Translations

- **C Standard**: C99, **Compilers**: CompCert (proven correct C compiler), clang, gcc
- **OS/platforms**: macOS, Linux, Windows, and others (opportunity-based)
- **Memory models**: static alloc. (done); ref-counting & full tracing-GC (*future*)
- **Platform Backends**
  - Conventional C applications running on Linux, Windows, macOS
  - SeL4 (part of Rockwell Collins, Adventium, Data61 team on DARPA CASE)
  - Experimental translations for…
    - Genode operating system framework
    - Minix 3 enhanced for separation (DHS CPSSec project)
    - FreeRTOS

Abstraction Levels - AADL State Machines

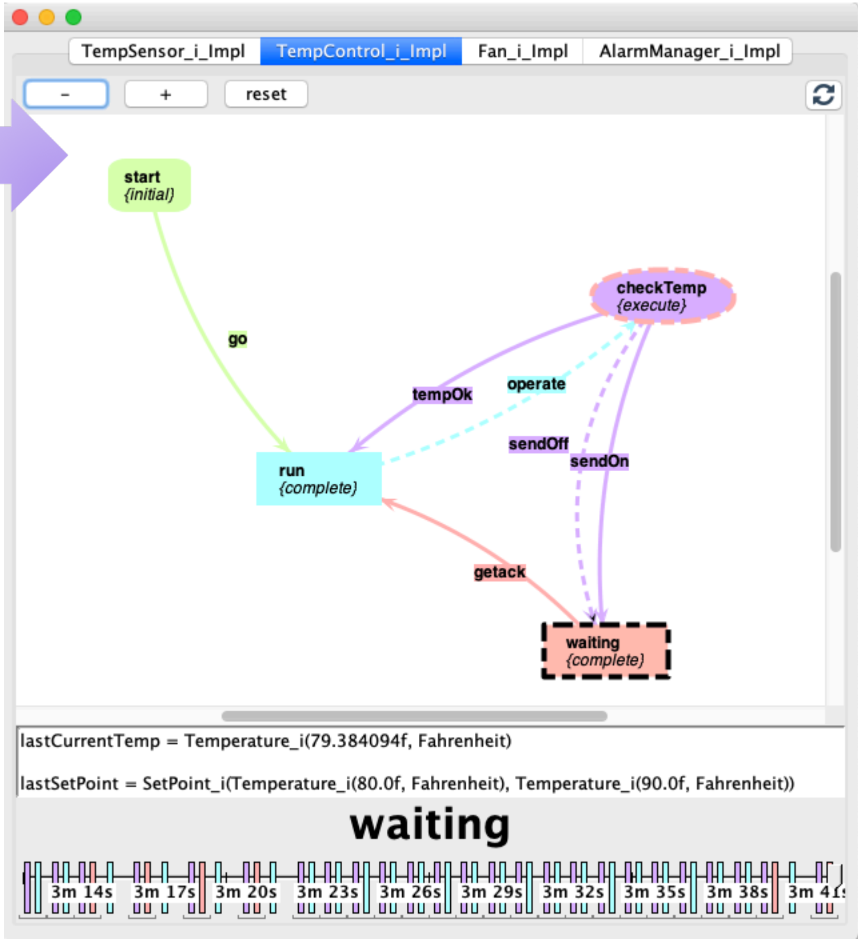
The simulation has a dynamic visualization of the BLESS/BA state machines of each AADL thread

AADL State Machine Specifications



Simulation

Compilation to, e.g., C

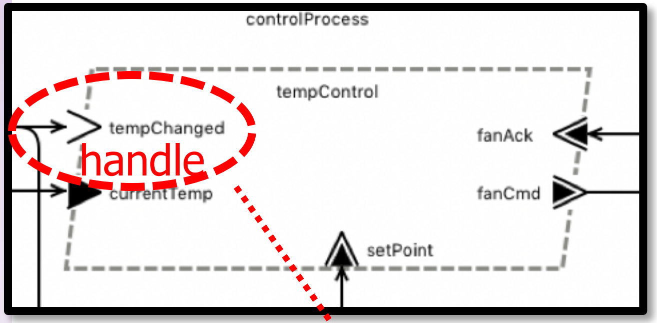


```
lastCurrentTemp = Temperature_i(79.384094f, Fahrenheit)

lastSetPoint = SetPoint_i(Temperature_i(80.0f, Fahrenheit), Temperature_i(90.0f, Fahrenheit))
```

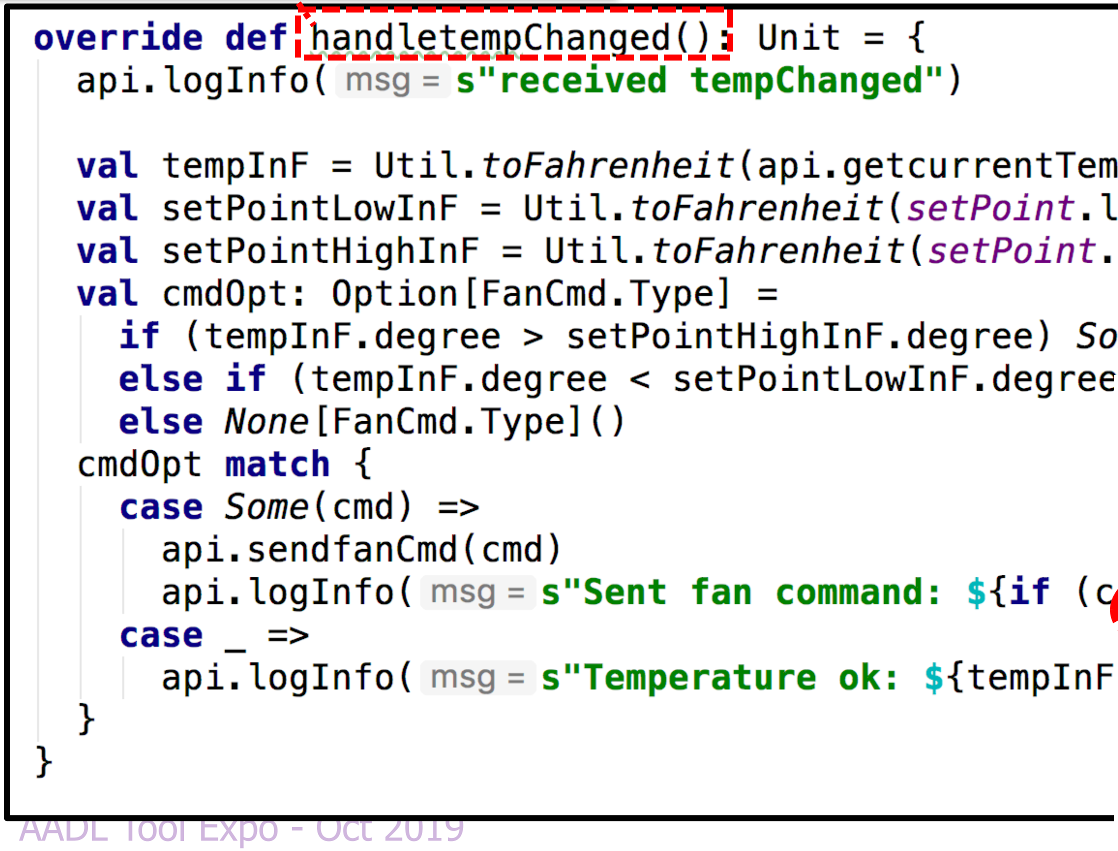
waiting

Army SBIR "GUMBO"
Adventium/KSU

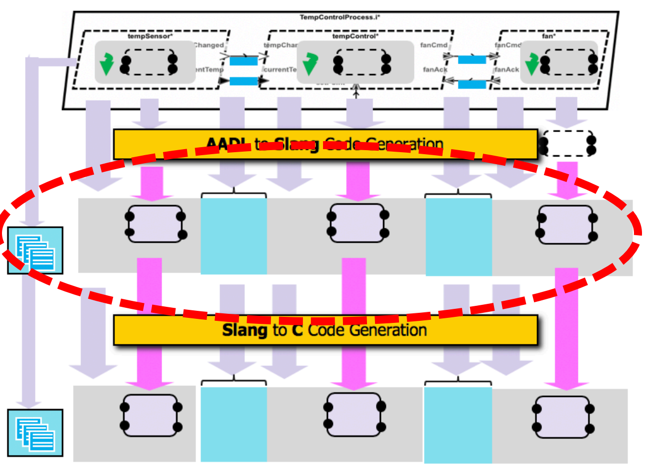
Component Implementations in Slang



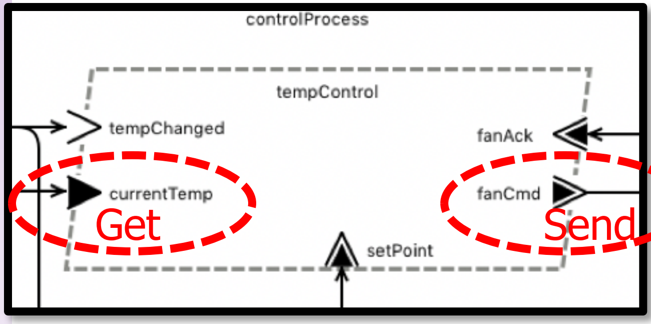
…Slang can be used to implement component business logic (corresponding to event handlers for incoming interface events)



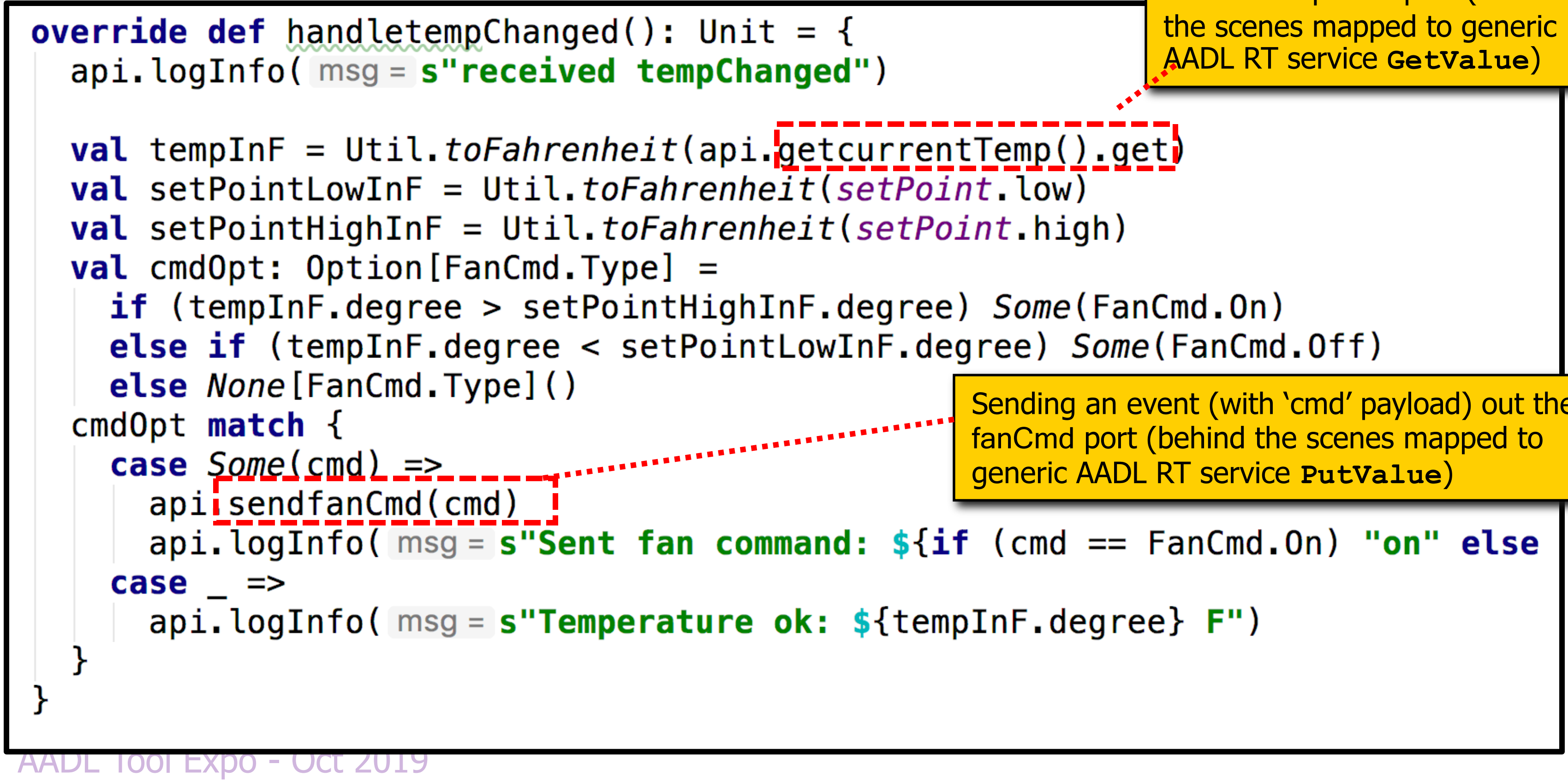
```
override def handletempChanged(): Unit = {
  api.logInfo( msg = s"received tempChanged")

  val tempInF = Util.toFahrenheit(api.getcurrentTemp().get)
  val setPointLowInF = Util.toFahrenheit(setPoint.low)
  val setPointHighInF = Util.toFahrenheit(setPoint.high)
  val cmdOpt: Option[FanCmd.Type] =
    if (tempInF.degree > setPointHighInF.degree) Some(FanCmd.On)
    else if (tempInF.degree < setPointLowInF.degree) Some(FanCmd.Off)
    else None[FanCmd.Type]()
  cmdOpt match {
    case Some(cmd) =>
      api.sendfanCmd(cmd)
      api.logInfo( msg = s"Sent fan command: ${if (c
    case _ =>
      api.logInfo( msg = s"Temperature ok: ${tempInF
  }
}
```



Component Implementations in Slang



…Slang implementations include calls to publish events on output ports and get/set values of data ports



```
override def handletempChanged(): Unit = {
  api.logInfo( msg = s"received tempChanged")

  val tempInF = Util.toFahrenheit(api.getcurrentTemp().get)
  val setPointLowInF = Util.toFahrenheit(setPoint.low)
  val setPointHighInF = Util.toFahrenheit(setPoint.high)
  val cmdOpt: Option[FanCmd.Type] =
    if (tempInF.degree > setPointHighInF.degree) Some(FanCmd.On)
    else if (tempInF.degree < setPointLowInF.degree) Some(FanCmd.Off)
    else None[FanCmd.Type]()
  cmdOpt match {
    case Some(cmd) =>
      api.sendfanCmd(cmd)
      api.logInfo( msg = s"Sent fan command: ${if (cmd == FanCmd.On) "on" else
    case _ =>
      api.logInfo( msg = s"Temperature ok: ${tempInF.degree} F")
  }
}
```

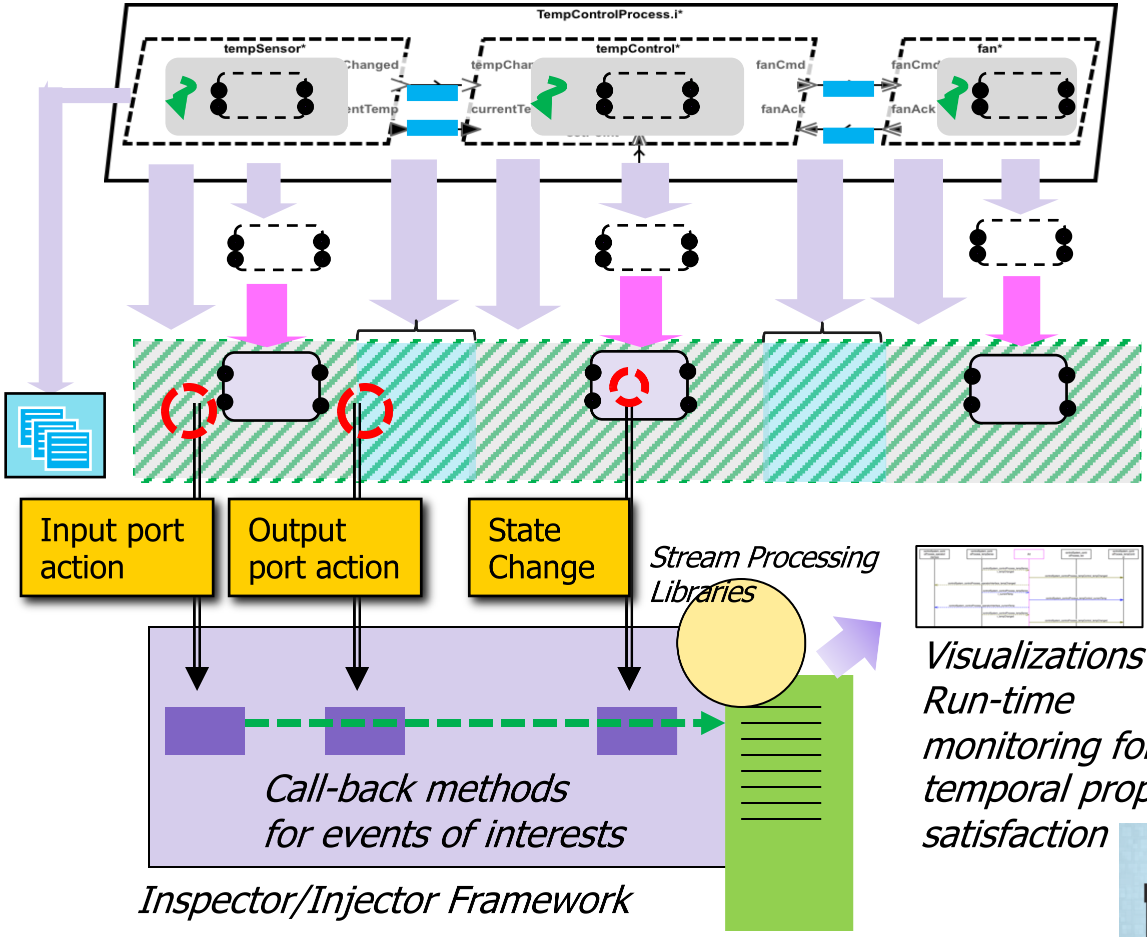
Reading a value from the currentTemp data port (behind the scenes mapped to generic AADL RT service **`GetValue`**)

Sending an event (with 'cmd' payload) out the fanCmd port (behind the scenes mapped to generic AADL RT service **`PutValue`**)

SEI ISSE FY'19

HAMR Run-time Monitoring & Visualization

The HAMR Debugging infrastructure provides hooks for registering **call-back methods** that get invoked where there is an **action on an output port or input port**, or when the value of a application **component local variable changes**.



Input port action

Output port action

State Change

Call-back methods for events of interests

System Modeling and Analysis

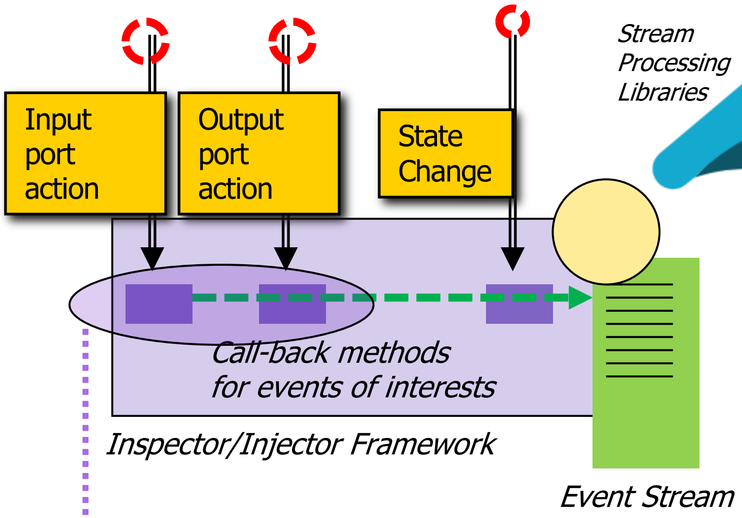
...in AADL

Tapping into the Slang **Reference Implementation** for execution **events** and **state changes** to drive **run-time monitoring**

Visualizations & Run-time monitoring for temporal property satisfaction

Event Stream

Example Event Stream Filtering



Input port action

Output port action

State Change

Call-back methods for events of interests

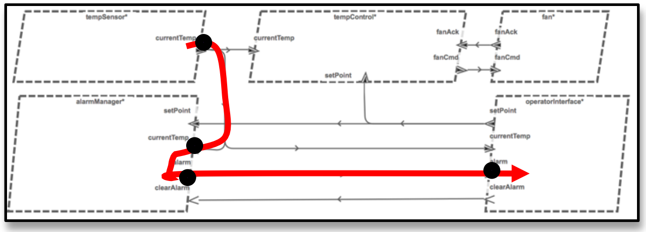
Inspector/Injector Framework

Event Stream

Stream Processing Libraries

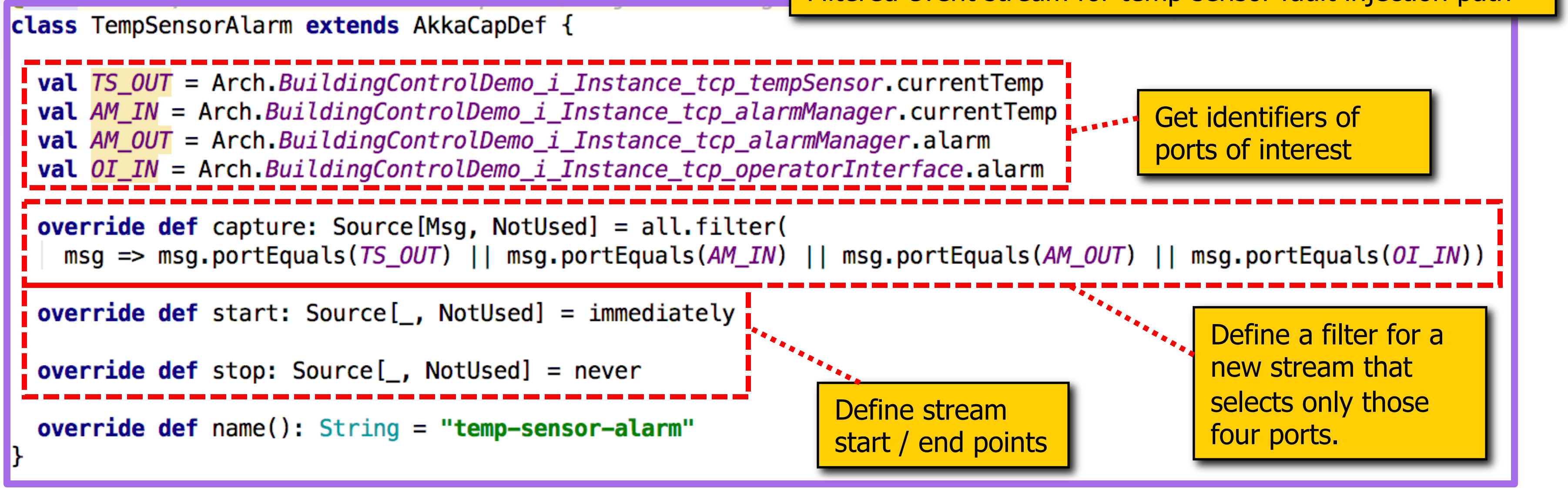
akka

https://akka.io



I'd like to visualize events on the temp sensor fault mitigation path

Filtered event stream for temp sensor fault injection path



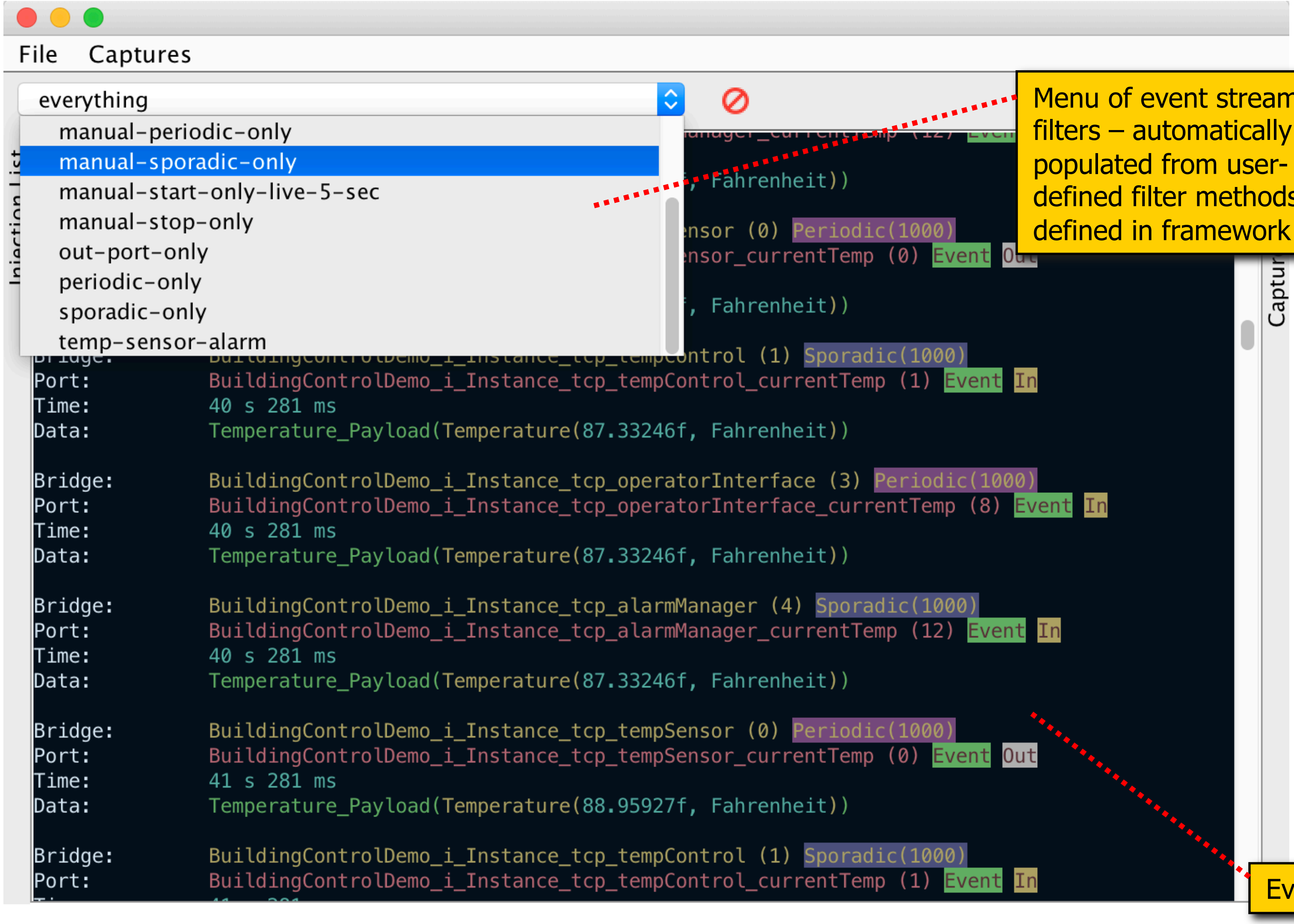
```
class TempSensorAlarm extends AkkaCapDef {

  val TS_OUT = Arch.BuildingControlDemo_i_Instance_tcp_tempSensor.currentTemp
  val AM_IN = Arch.BuildingControlDemo_i_Instance_tcp_alarmManager.currentTemp
  val AM_OUT = Arch.BuildingControlDemo_i_Instance_tcp_alarmManager.alarm
  val OI_IN = Arch.BuildingControlDemo_i_Instance_tcp_operatorInterface.alarm

  override def capture: Source[Msg, NotUsed] = all.filter(
    msg => msg.portEquals(TS_OUT) || msg.portEquals(AM_IN) || msg.portEquals(AM_OUT) || msg.portEquals(OI_IN))

  override def start: Source[_, NotUsed] = immediately

  override def stop: Source[_, NotUsed] = never

  override def name(): String = "temp-sensor-alarm"
}
```

Get identifiers of ports of interest

Define a filter for a new stream that selects only those four ports.

Define stream start / end points

Event Filtering

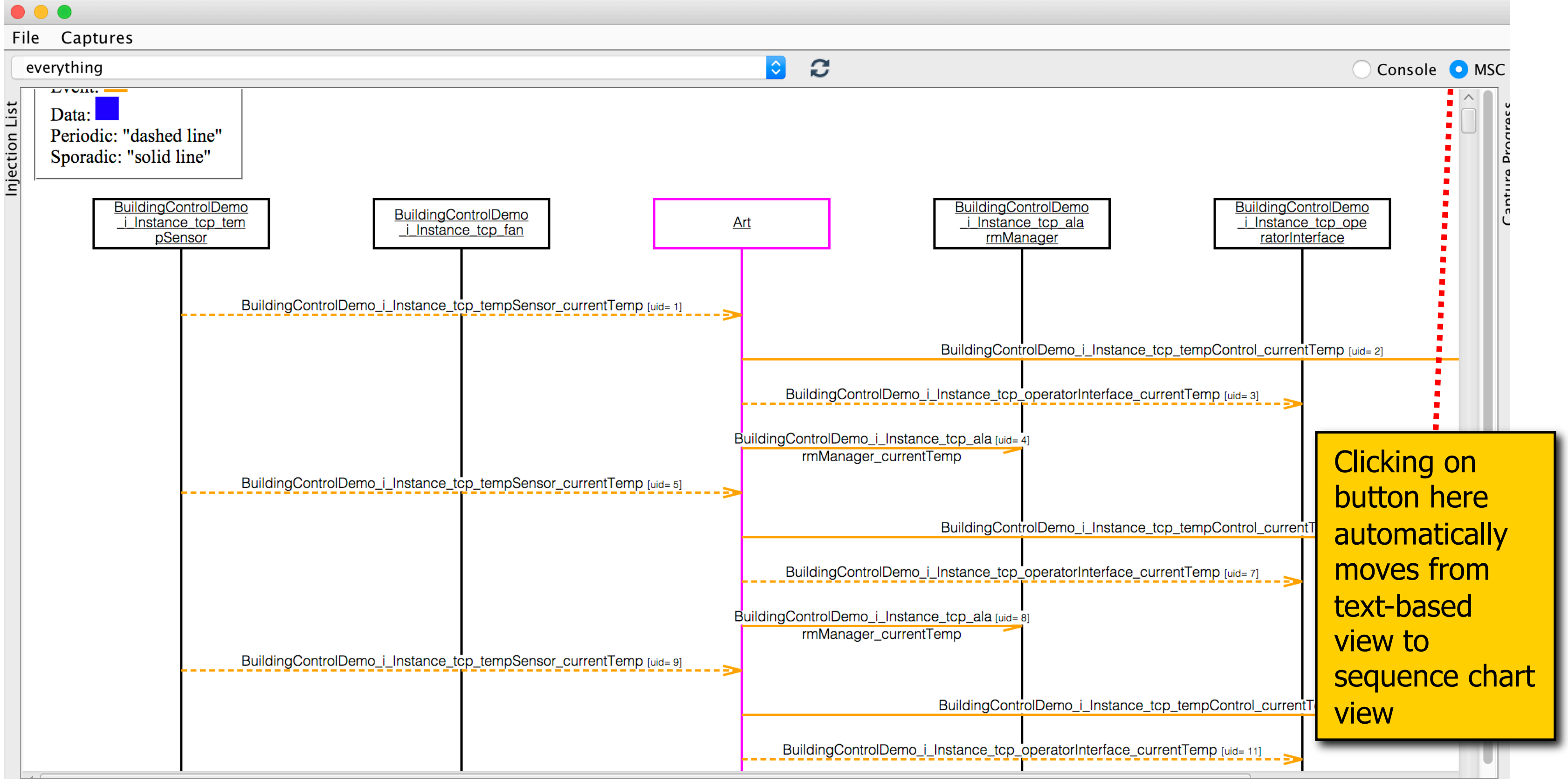


```
                anager_currentTemp (12) Even
                f, Fahrenheit))

                nsor (0) Periodic(1000)
                nsor_currentTemp (0) Event Out
                f, Fahrenheit))

Bridge:         ontrol (1) Sporadic(1000)
Port:           BuildingControlDemo_i_Instance_tcp_tempControl_currentTemp (1) Event In
Time:           40 s 281 ms
Data:           Temperature_Payload(Temperature(87.33246f, Fahrenheit))

Bridge:         BuildingControlDemo_i_Instance_tcp_operatorInterface (3) Periodic(1000)
Port:           BuildingControlDemo_i_Instance_tcp_operatorInterface_currentTemp (8) Event In
Time:           40 s 281 ms
Data:           Temperature_Payload(Temperature(87.33246f, Fahrenheit))

Bridge:         BuildingControlDemo_i_Instance_tcp_alarmManager (4) Sporadic(1000)
Port:           BuildingControlDemo_i_Instance_tcp_alarmManager_currentTemp (12) Event In
Time:           40 s 281 ms
Data:           Temperature_Payload(Temperature(87.33246f, Fahrenheit))

Bridge:         BuildingControlDemo_i_Instance_tcp_tempSensor (0) Periodic(1000)
Port:           BuildingControlDemo_i_Instance_tcp_tempSensor_currentTemp (0) Event Out
Time:           41 s 281 ms
Data:           Temperature_Payload(Temperature(88.95927f, Fahrenheit))

Bridge:         BuildingControlDemo_i_Instance_tcp_tempControl (1) Sporadic(1000)
Port:           BuildingControlDemo_i_Instance_tcp_tempControl_currentTemp (1) Event In
```

Menu of event stream filters – automatically populated from user-defined filter methods defined in framework

Event stream

Auto-generated Sequence Chart Visualization

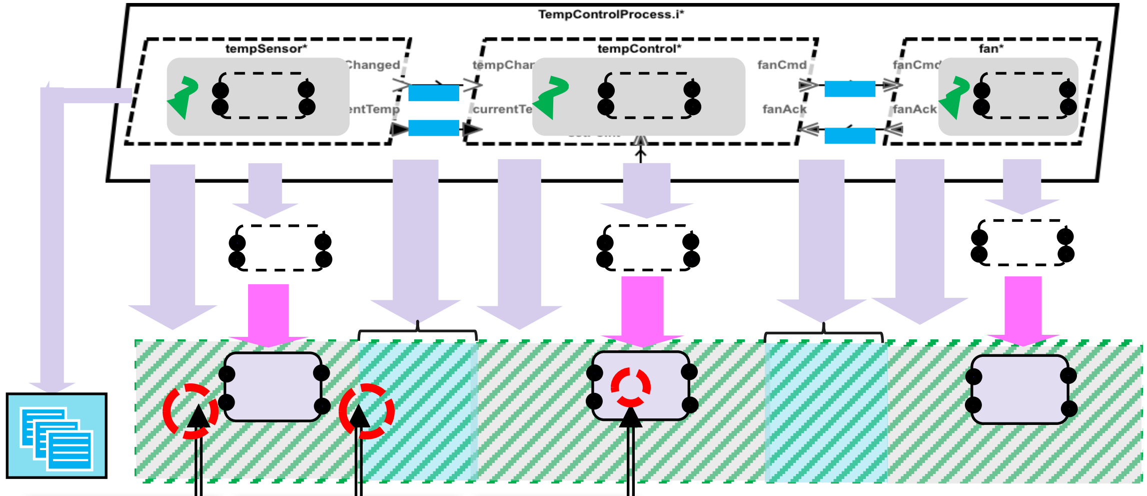

File
Captures
everything
○ Console
● MSC
Injection List
Capture Progress
Event:
Data: 🟦
Periodic: "dashed line"
Sporadic: "solid line"
BuildingControlDemo_i_Instance_tcp_tempSensor
BuildingControlDemo_i_Instance_tcp_fan
Art
BuildingControlDemo_i_Instance_tcp_alarmManager
BuildingControlDemo_i_Instance_tcp_operatorInterface
BuildingControlDemo_i_Instance_tcp_tempSensor_currentTemp [uid= 1]
BuildingControlDemo_i_Instance_tcp_tempControl_currentTemp [uid= 2]
BuildingControlDemo_i_Instance_tcp_operatorInterface_currentTemp [uid= 3]
BuildingControlDemo_i_Instance_tcp_alarmManager_currentTemp [uid= 4]
BuildingControlDemo_i_Instance_tcp_tempSensor_currentTemp [uid= 5]
BuildingControlDemo_i_Instance_tcp_tempControl_currentT
BuildingControlDemo_i_Instance_tcp_operatorInterface_currentTemp [uid= 7]
BuildingControlDemo_i_Instance_tcp_alarmManager_currentTemp [uid= 8]
BuildingControlDemo_i_Instance_tcp_tempSensor_currentTemp [uid= 9]
BuildingControlDemo_i_Instance_tcp_tempControl_currentT
BuildingControlDemo_i_Instance_tcp_operatorInterface_currentTemp [uid= 11]

Clicking on button here automatically moves from text-based view to sequence chart view

SEI ISSE FY'19

HAMR Fault Injection and Testing

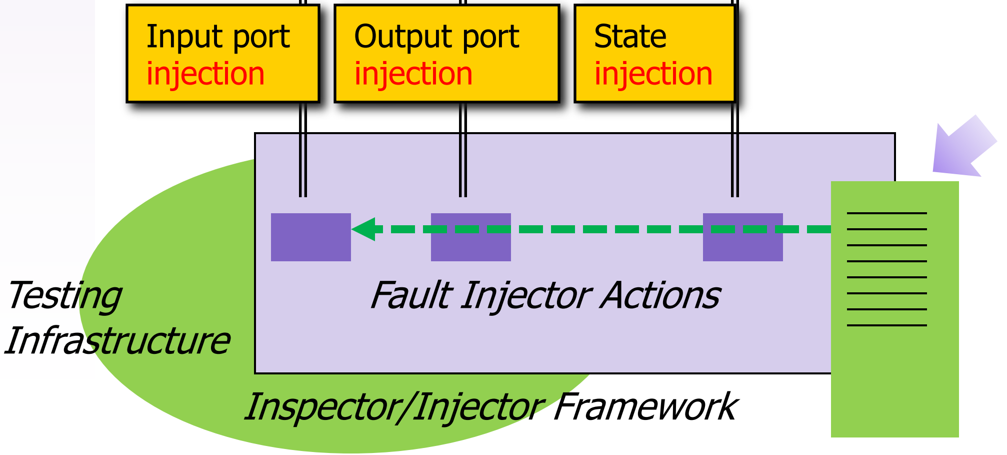
The HAMR Debugging infrastructure allows one to **inject values at an output port or input port**. It also allows a **component local variable to be directly set/perturbed.**



System Modeling and Analysis

...in AADL

Injecting faults into the Slang **Reference Implementation**



Input port injection

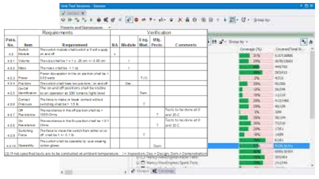
Output port injection

State injection

Fault Injector Actions

Testing Infrastructure

Inspector/Injector Framework

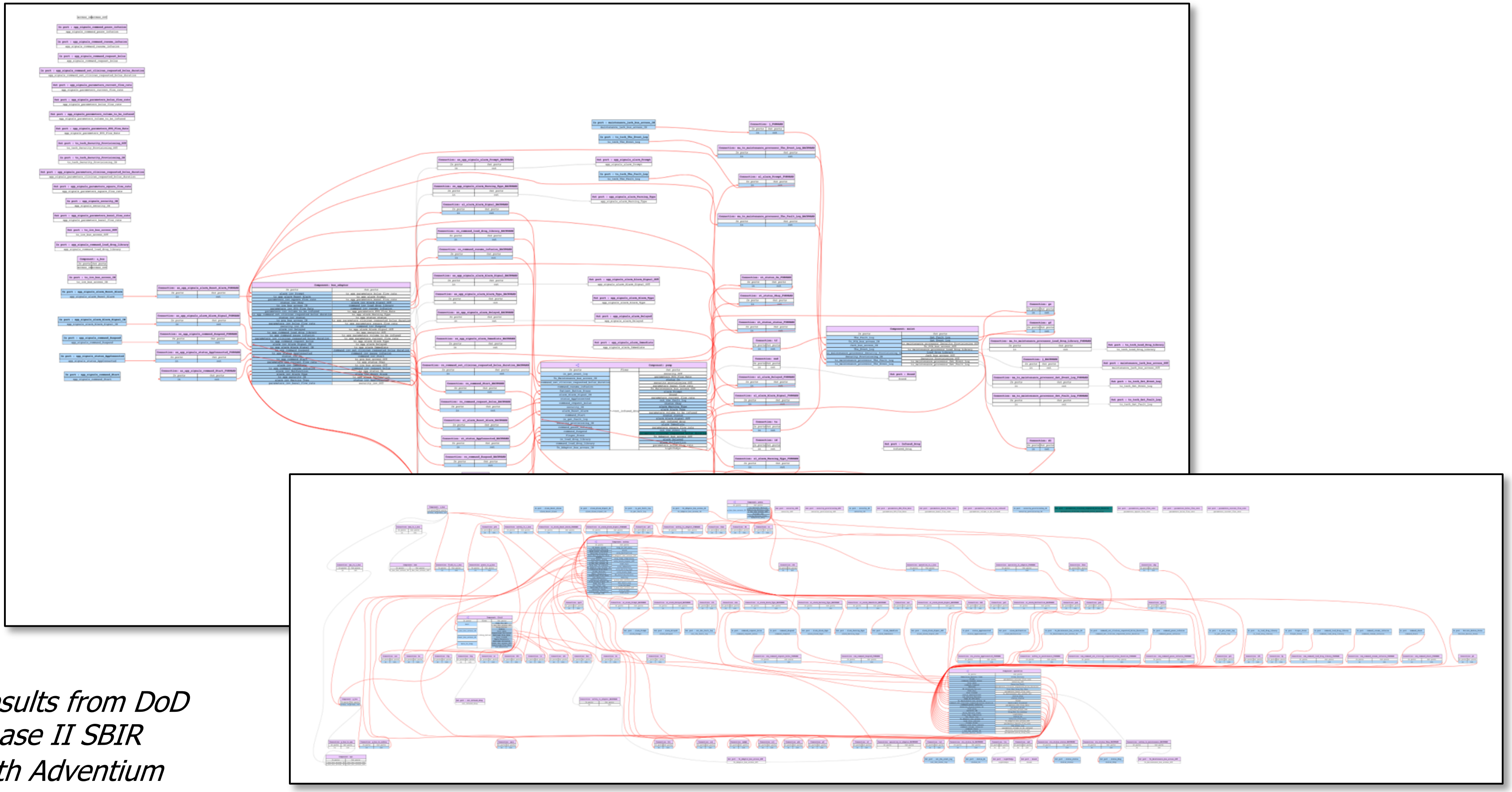


Requirements, Hazard Analysis, and Test Plans

Fault Injection Scripts

Flow, Dependence, and Error Propagation Visualization & Querying

The KSU Awas tool builds scalable interactive visualizations of AADL information flows and error propagations

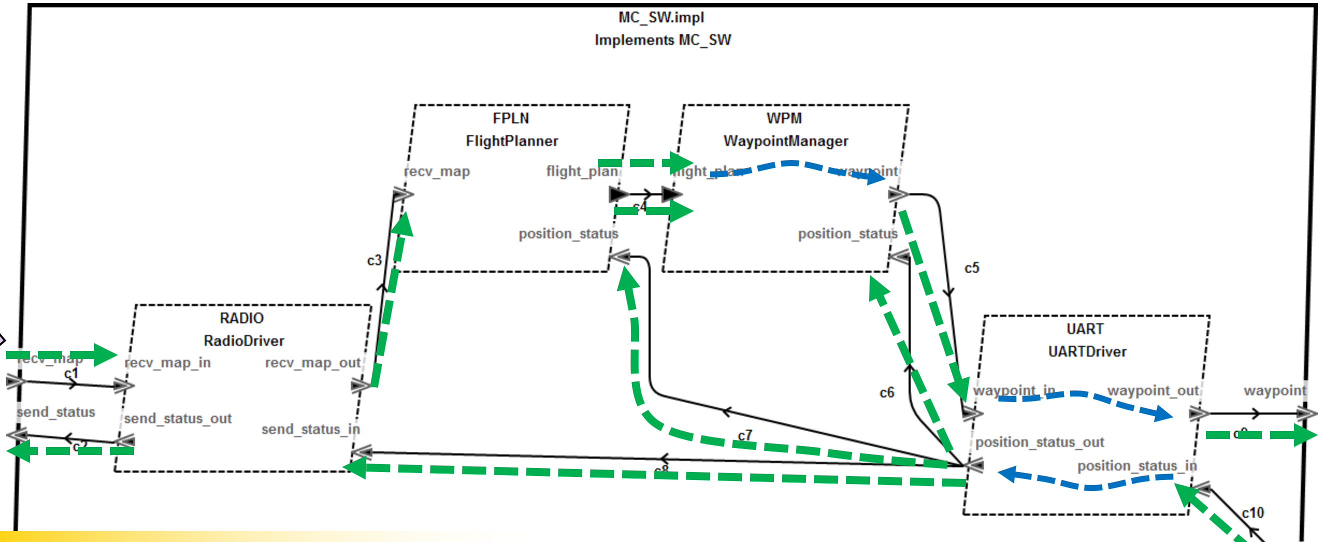


Results from DoD Phase II SBIR with Adventium Labs

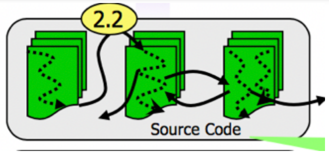
Information flow graphs can be dynamically browsed and queried with path logic.

Information Flow Analysis Foundation

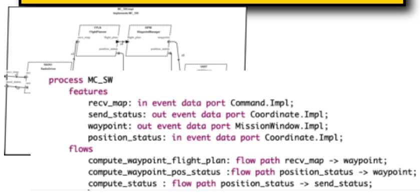
Internal dependency graphs upon which analysis is performed are built from architecture connections and intra-component flows as well as EMv2 annotations



Markup/Interaction on source code (future – based on past work)

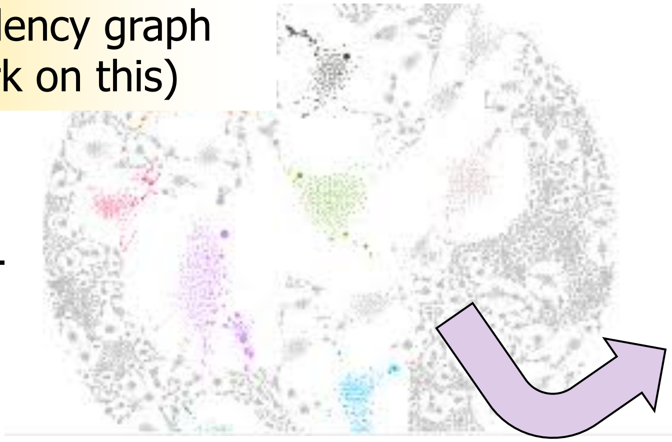


Markup/Interaction on AADL Artifacts (future – based on past work)

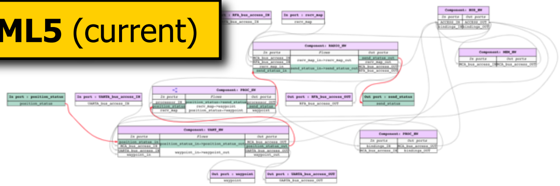


```
process MC_SW
    features
        recv_map: in event data port Command.Impl;
        send_status: out event data port Coordinate.Impl;
        waypoint: out event data port MissionWindow.Impl;
        position_status: in event data port Coordinate.Impl;
    flows
        compute_waypoint_flight_plan: flow path recv_map -> waypoint;
        compute_waypoint_pos_status :flow path position_status -> waypoint;
        compute_status : flow path position_status -> send_status;
```

Internal dependency graph (algorithms work on this)

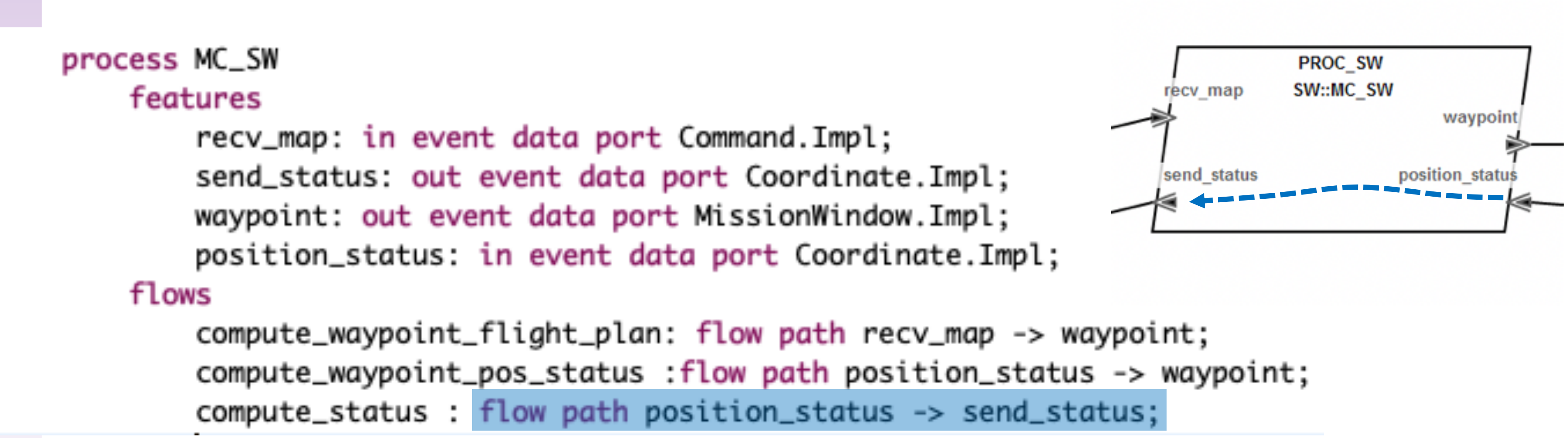


HTML5 (current)



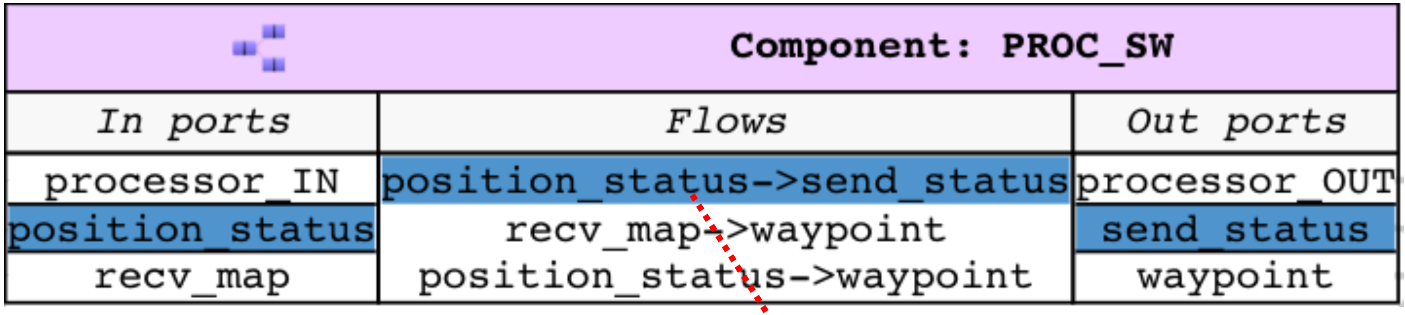
Interactions and rendered results

Details of Information Flow Rendering



```
process MC_SW
    features
        recv_map: in event data port Command.Impl;
        send_status: out event data port Coordinate.Impl;
        waypoint: out event data port MissionWindow.Impl;
        position_status: in event data port Coordinate.Impl;
    flows
        compute_waypoint_flight_plan: flow path recv_map -> waypoint;
        compute_waypoint_pos_status :flow path position_status -> waypoint;
        compute_status : flow path position_status -> send_status;
```

PROC_SW
SW::MC_SW
recv_map
waypoint
send_status
position_status

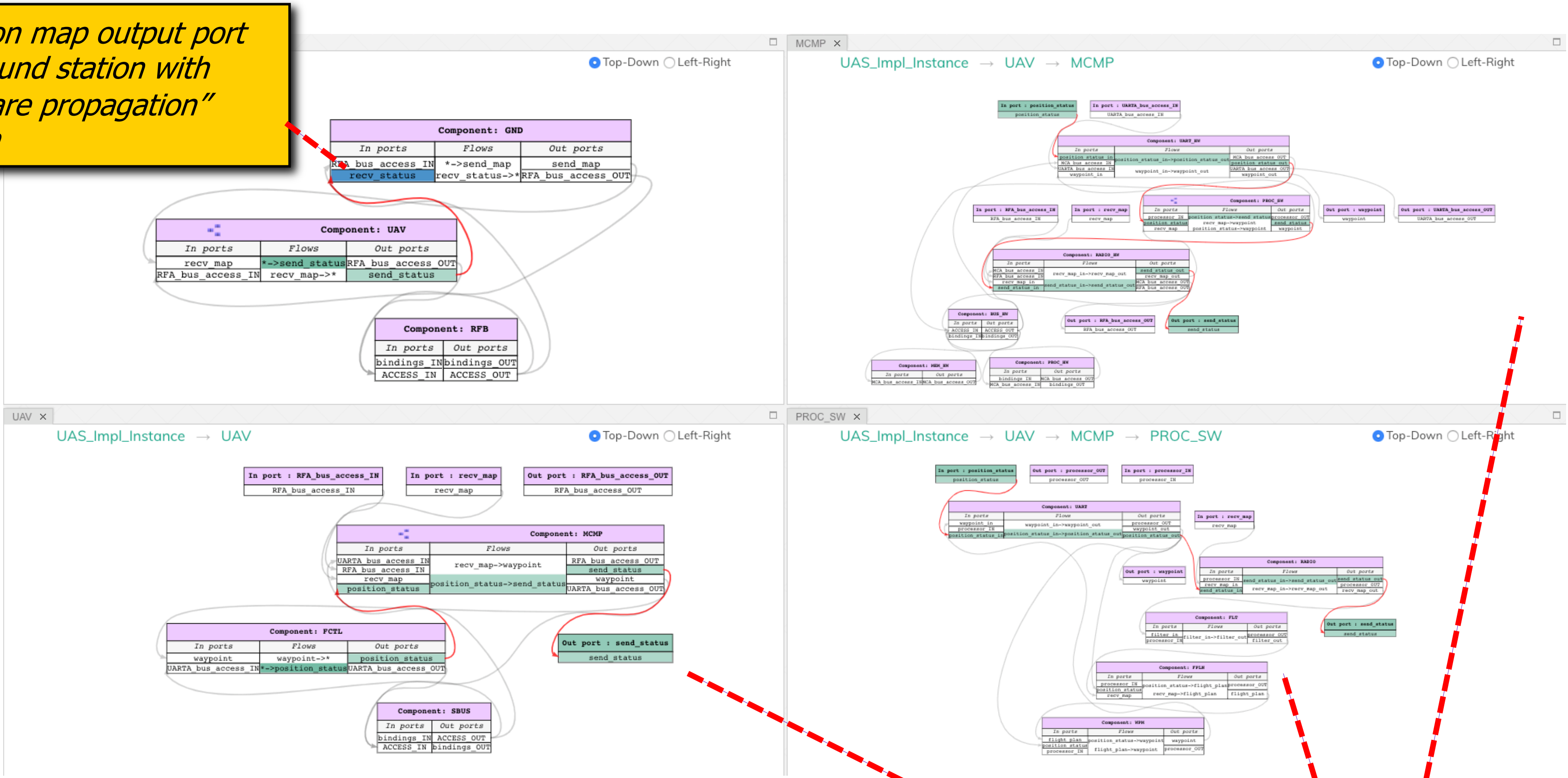


Component: PROC_SW		
In ports	*Flows*	*Out ports*
processor_IN	position_status->send_status	processor_OUT
position_status	recv_map->waypoint	send_status
recv_map	position_status->waypoint	waypoint

Flows: In this case, intra-component flows are not sources and sinks, but **flows of information between inputs and outputs.**

Interactive Browsing of Information Flows

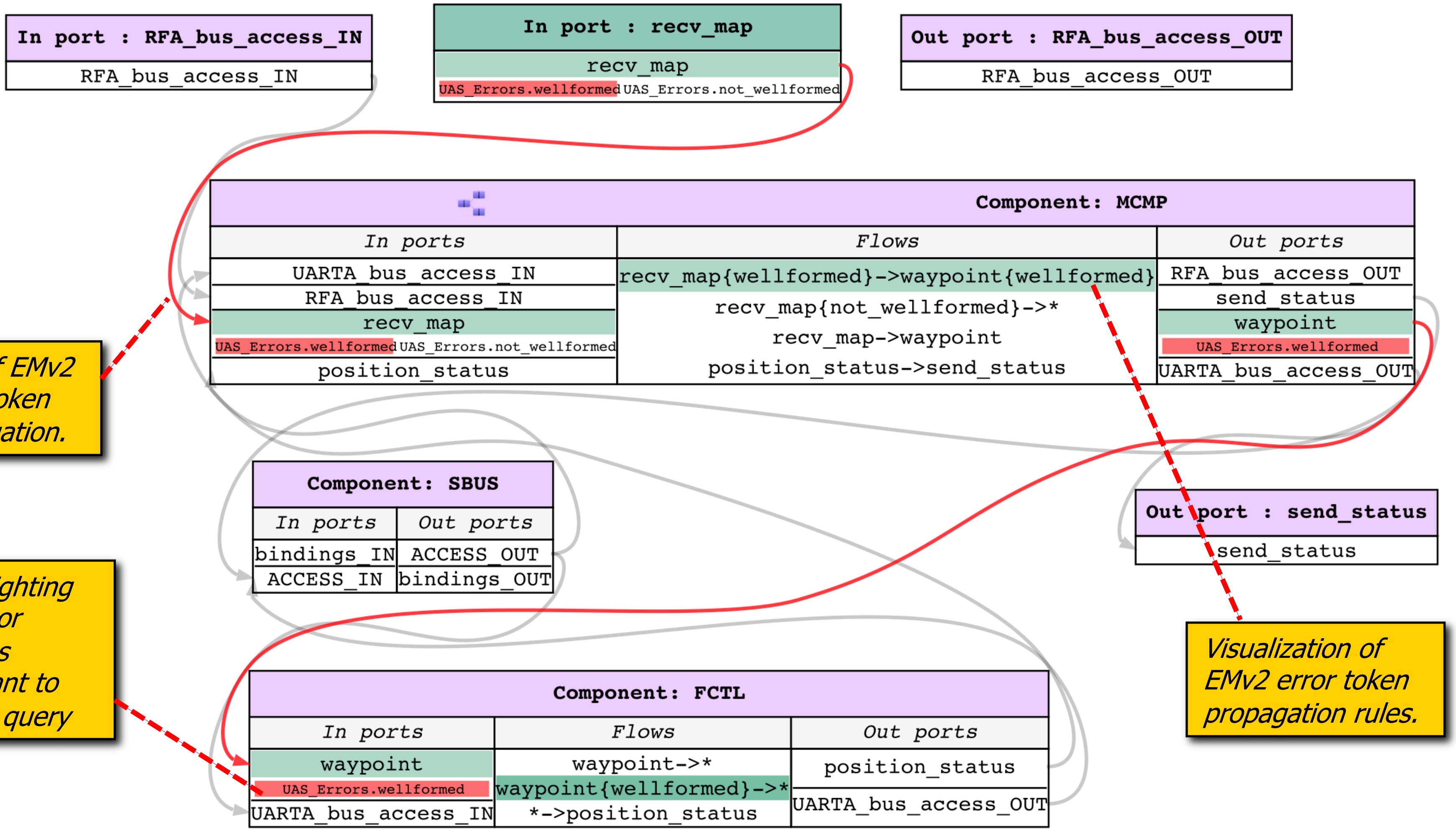
Example: In Ground Station / UAV example used on DARPA CASE, ask “how does map information propagation from ground station to UAV and through UAV’s mission computer to produce a waypoint?”



Click on map output port of ground station with “forware propagation” option

Immediately see results of across different subsystems.

Example Representation of AADL EMv2 Error Propagation (Hazard Analysis)



In port : RFA_bus_access_IN
RFA_bus_access_IN

In port : recv_map	
recv_map	
UAS_Errors.wellformed	UAS_Errors.not_wellformed

Out port : RFA_bus_access_OUT
RFA_bus_access_OUT

Component: MCMP

In ports	Flows	Out ports
UARTA_bus_access_IN	recv_map{wellformed}->waypoint{wellformed}	RFA_bus_access_OUT
RFA_bus_access_IN	recv_map{not_wellformed}->*	send_status
recv_map	recv_map->waypoint	waypoint
UAS_Errors.wellformed UAS_Errors.not_wellformed		UAS_Errors.wellformed
position_status	position_status->send_status	UARTA_bus_access_OUT

Component: SBUS

In ports	Out ports
bindings_IN	ACCESS_OUT
ACCESS_IN	bindings_OUT

Out port : send_status
send_status

Component: FCTL

In ports	Flows	Out ports
waypoint	waypoint->*	position_status
UAS_Errors.wellformed	waypoint{wellformed}->*	UARTA_bus_access_OUT
UARTA_bus_access_IN	*->position_status	

Path of EMv2 error token propagation.

Highlighting of error tokens relevant to given query

Visualization of EMv2 error token propagation rules.

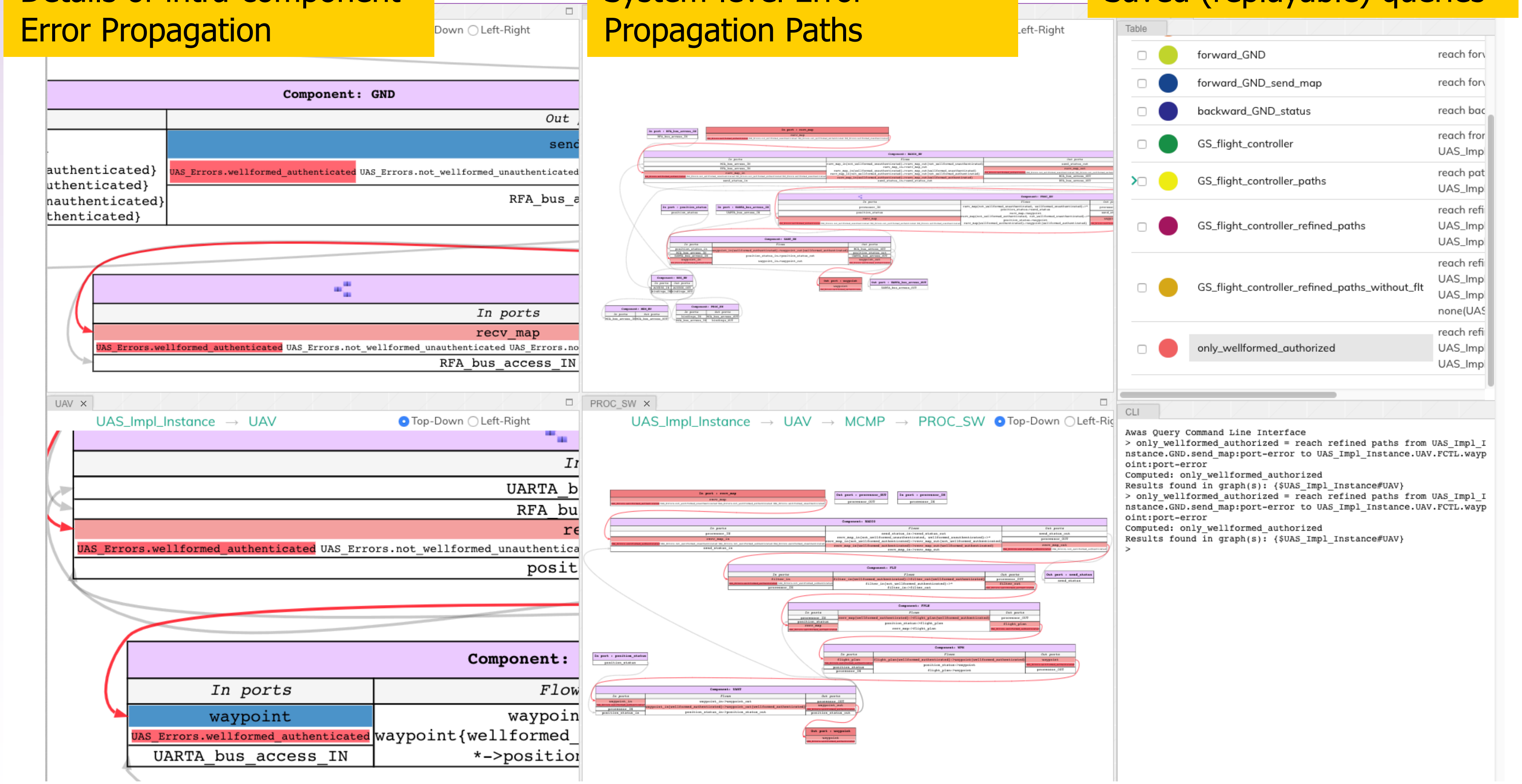
In essence, capturing a "causality chain" in hazard analysis (e.g. FMEA, STPA)

Example Visualization of AADL EMv2 Error Propagation (Hazard Analysis)

Details of intra-component Error Propagation

System-level Error Propagation Paths

Saved (replayable) queries



□	●	forward_GND	reach forv	
□	●	forward_GND_send_map	reach forv	
□	●	backward_GND_status	reach bac	
□	●	GS_flight_controller	reach fror UAS_Impl	
› □	●	GS_flight_controller_paths	reach pat UAS_Impl	
□	●	GS_flight_controller_refined_paths	reach refi UAS_Impl UAS_Impl	
□	●	GS_flight_controller_refined_paths_without_flt	reach refi UAS_Impl UAS_Impl none(UAS	
□	●	only_wellformed_authorized	reach refi UAS_Impl UAS_Impl	

```
Awas Query Command Line Interface
> only_wellformed_authorized = reach refined paths from UAS_Impl_I
nstance.GND.send_map:port-error to UAS_Impl_Instance.UAV.FCTL.wayp
oint:port-error
Computed: only_wellformed_authorized
Results found in graph(s): {$UAS_Impl_Instance#UAV}
> only_wellformed_authorized = reach refined paths from UAS_Impl_I
nstance.GND.send_map:port-error to UAS_Impl_Instance.UAV.FCTL.wayp
oint:port-error
Computed: only_wellformed_authorized
Results found in graph(s): {$UAS_Impl_Instance#UAV}
>
```

Conclusions

- HAMR – Flexible simulation and code generation framework for AADL – capable of supporting multiple languages / platforms
  - Continuing to expand platforms supported – let us know if you are interested
- Integrated analysis and automated verification capabilities (see demo)
  - Significant long-term emphasis on scalable formal verification and certification arguments
- Applied on DARPA CASE project to ensure cyber-resiliency using partitioning platforms (e.g., micro-kernels)
- Related demos…
  - Adventium Labs
  - BLESS – Brian Larson / Multitude