

The Slang Embedded Code Generation and Analysis Framework for AADL

AADL Tool Expo – February 14, 2019

John Hatcliff

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

Robby

Professor
Kansas State University

Brian Larson, Jason Belt,
Hariharan Thiagarajan

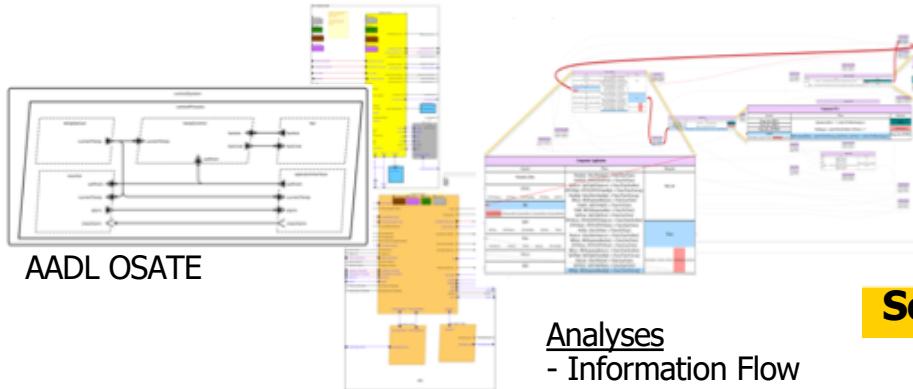
Research Associates
Kansas State University

In collaboration with Adventium Labs and SEI

Deeply Integrate Models and Programming Across Levels of Abstraction

System Modeling and Analysis (AADL)

Analysis and verification results moved up and down abstraction layers

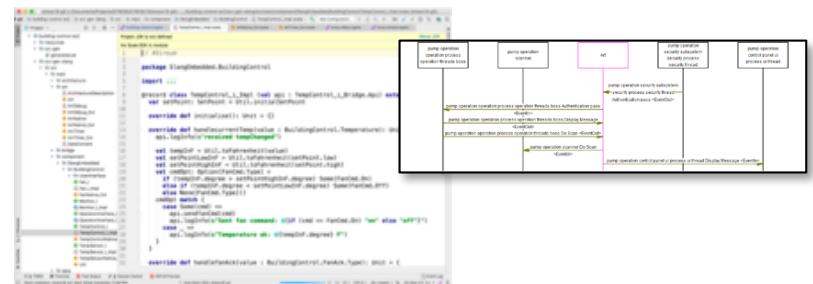


Analyses

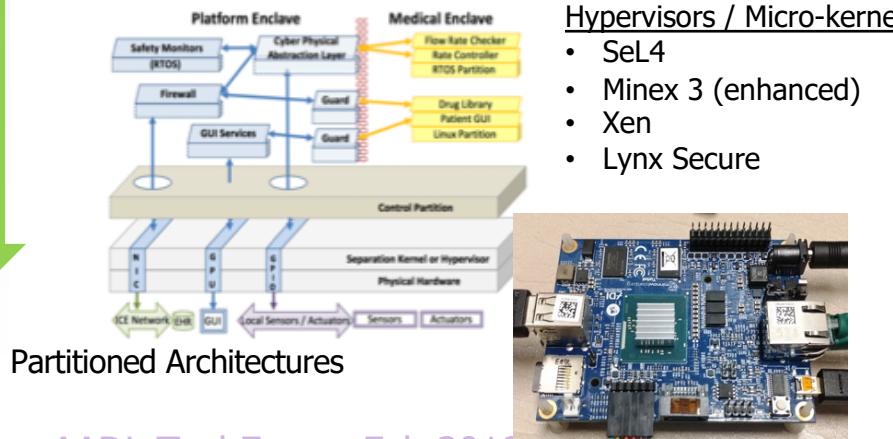
- Information Flow
- Functional Integration Constraints (component contracts)
- Scheduleability
- ...

Code Generation
-- Slang + Run-Time System Abstraction

Source Code, Simulation, Analysis, Verification



Deployment on Embedded/Distributed Platforms



AADL Tool Expo - Feb 2019

Slang – Subset of Scala for critical systems

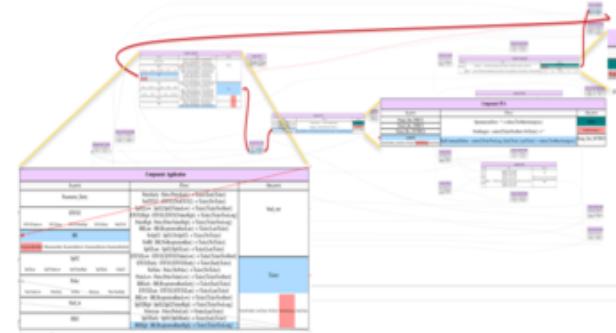
Code Generation, e.g.,

- C + Platform Run-Time System (primitives for controlling communication between partitions in a partitioning architecture)
- C compatible with CompCERT verified compiler
- Conforms to AADL Run-Time Services (informally specifies a computational model for **real-time threading** and **communication**).

AADL Tool Expo Focus

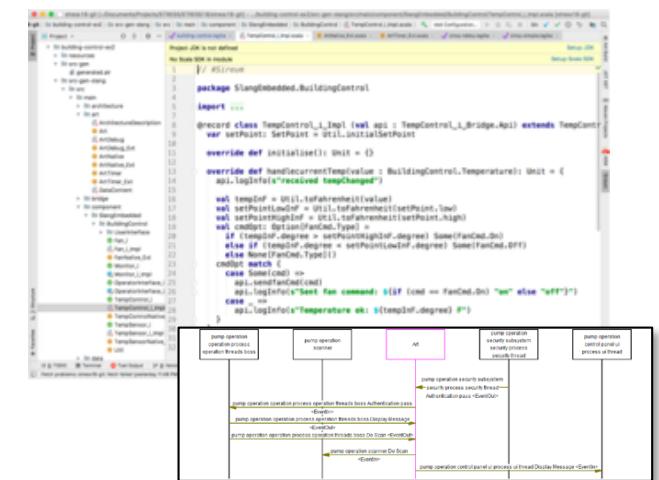
Information Flow and Error Propagation Analysis and Visualization

- Scalable visualizations
- Interactive browsing
- Queries for “What if..?” scenarios
- Policy Enforcement



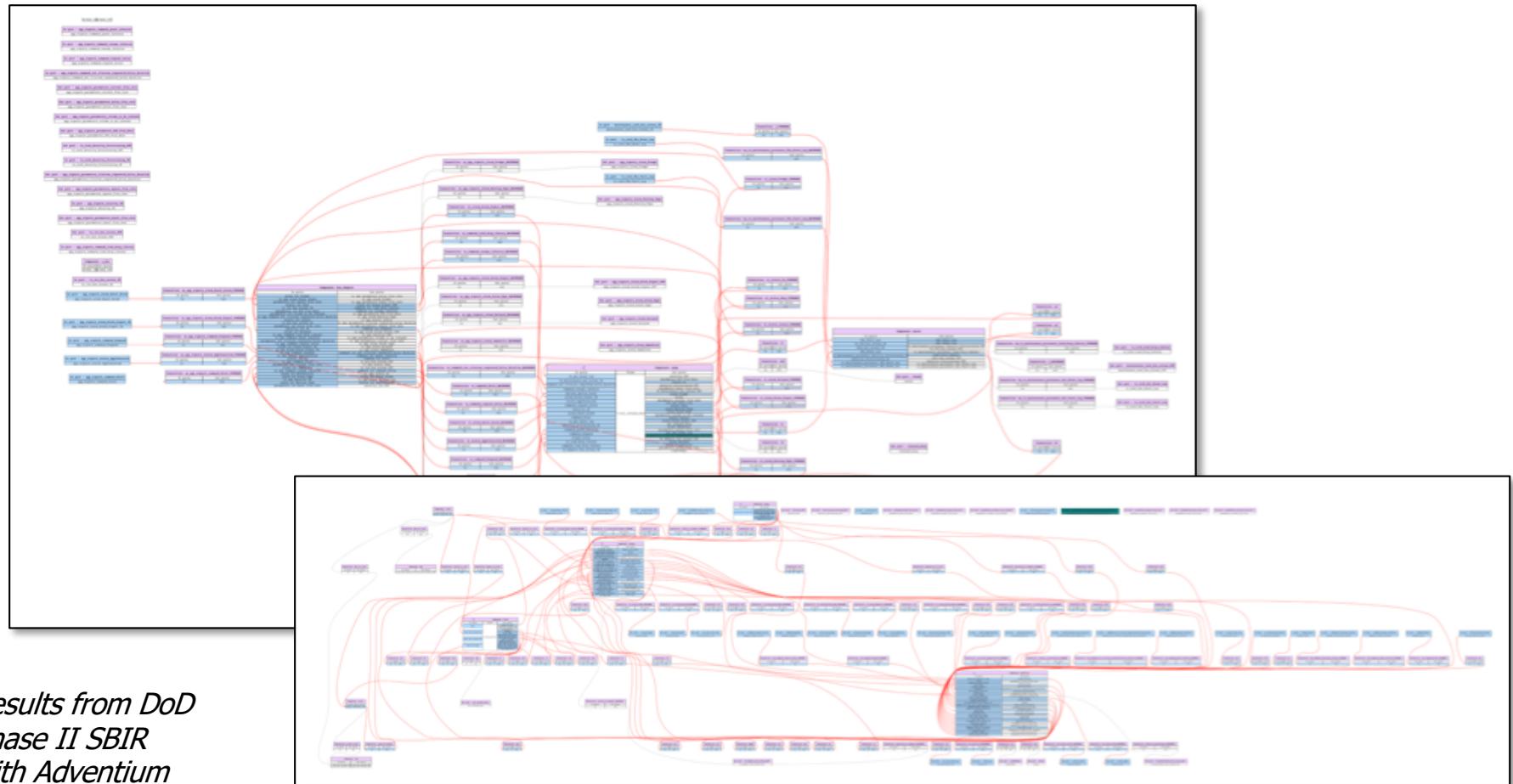
Code Generation, Application Development, Simulation/Testing

- Code generation to safety-critical subset of Scala (Slang) and then to C
- Flexible backend can support a variety of middleware and platform targets
- Special focus on applications on separation kernels (e.g., SeL4 – being used on DARPA CASE)



Flow, Dependence, and Error Propagation Visualization & Querying

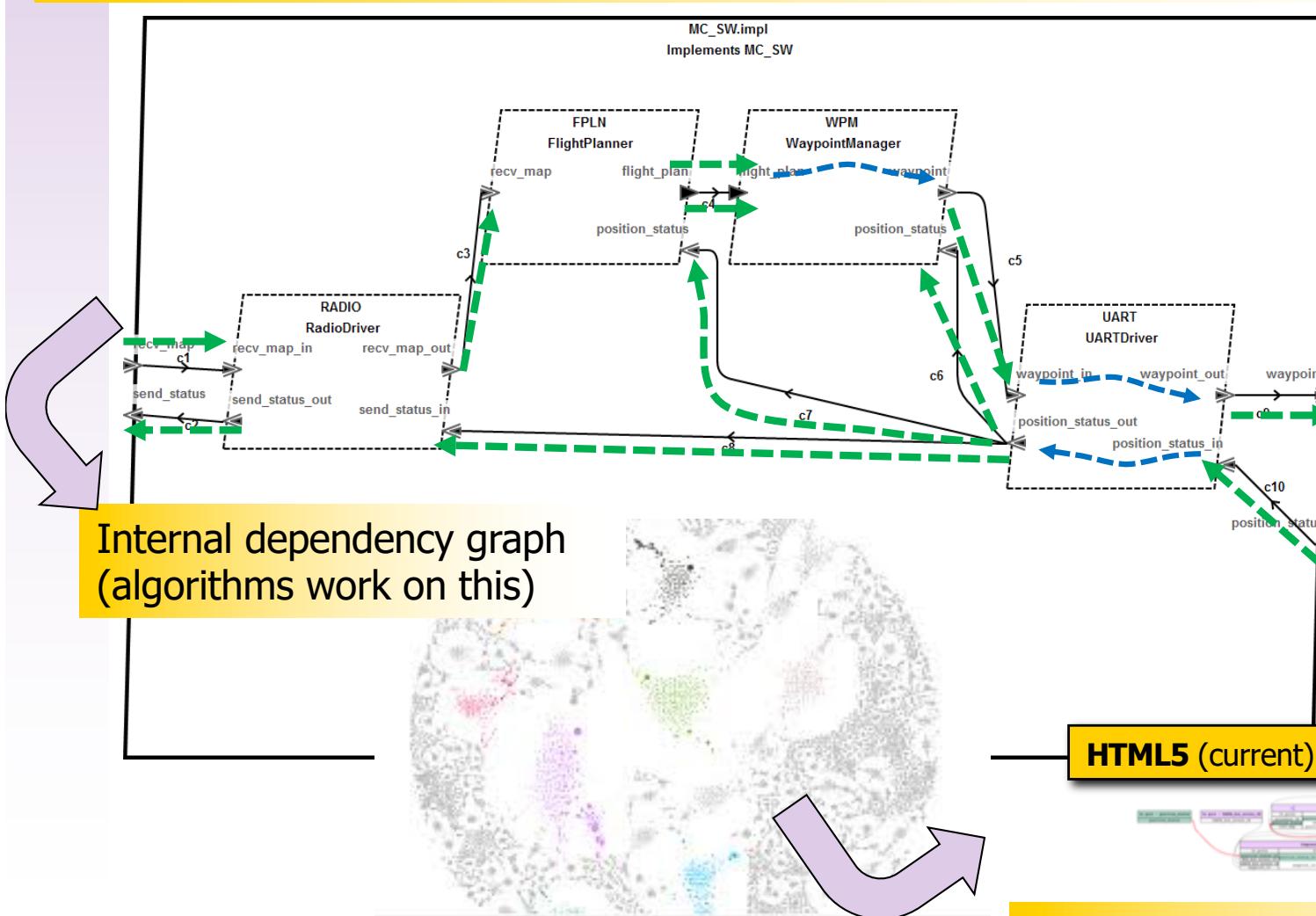
The KSU Awas tool builds scalable interactive visualizations of AADL information flows and error propagations



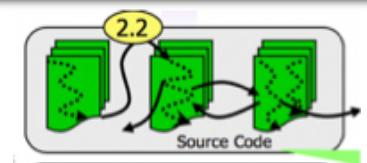
*Results from DoD
Phase II SBIR
with Adventium
Labs*

Information Flow Analysis Foundation

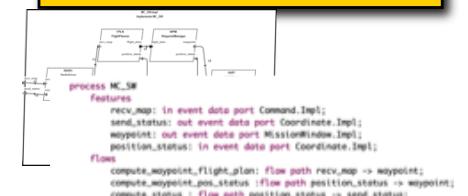
Internal dependency graphs upon which analysis is performed are built from **architecture connections** and **intra-component flows** as well as EMv2 annotations



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



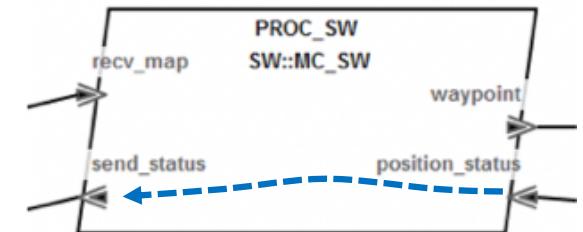
Markup/Interaction on AADL Artifacts
(future – based on past work)



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;
```



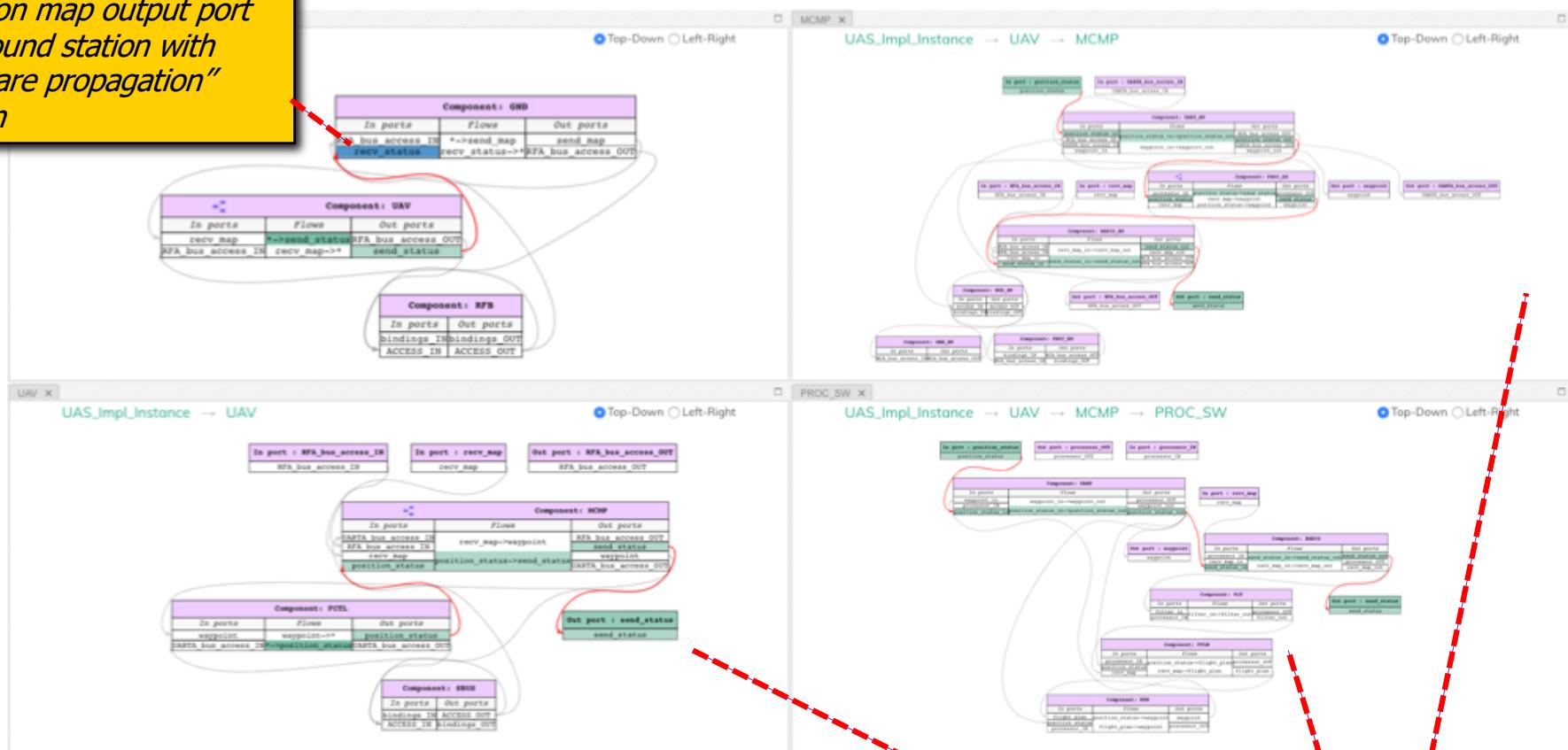
Component: PROC_SW		
In ports	Flows	Out ports
processor_IN	position_status->send_status	processor_OUT
position_status	recv_map->waypoint	send_status
recv_map	position_status->waypoint	waypoint

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

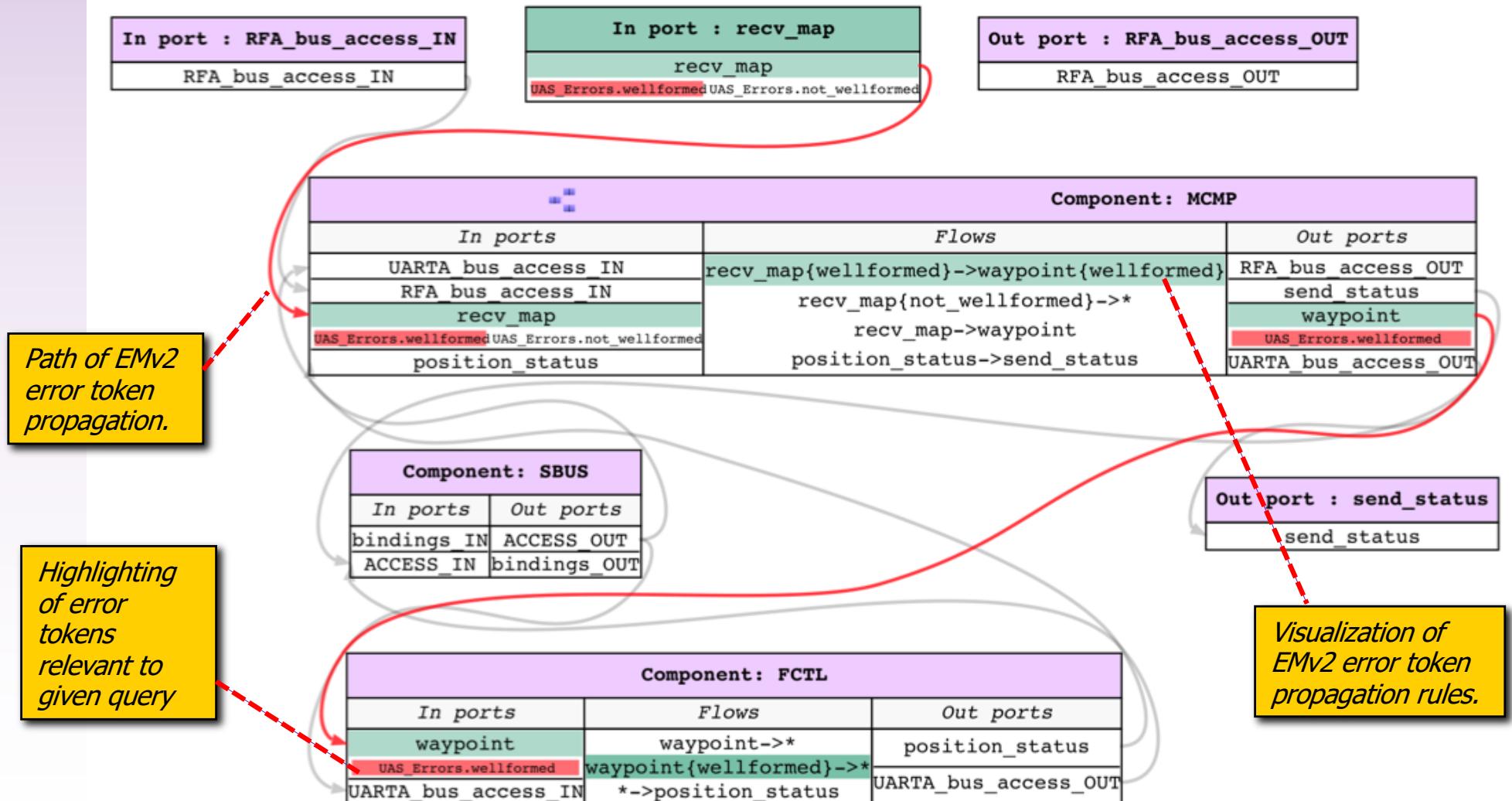
Example: In Ground Station / UAV example used on DARPA CASE, ask “how does map information propagation from ground station to UAV and through UAV’s mission computer to produce a waypoint?”

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)

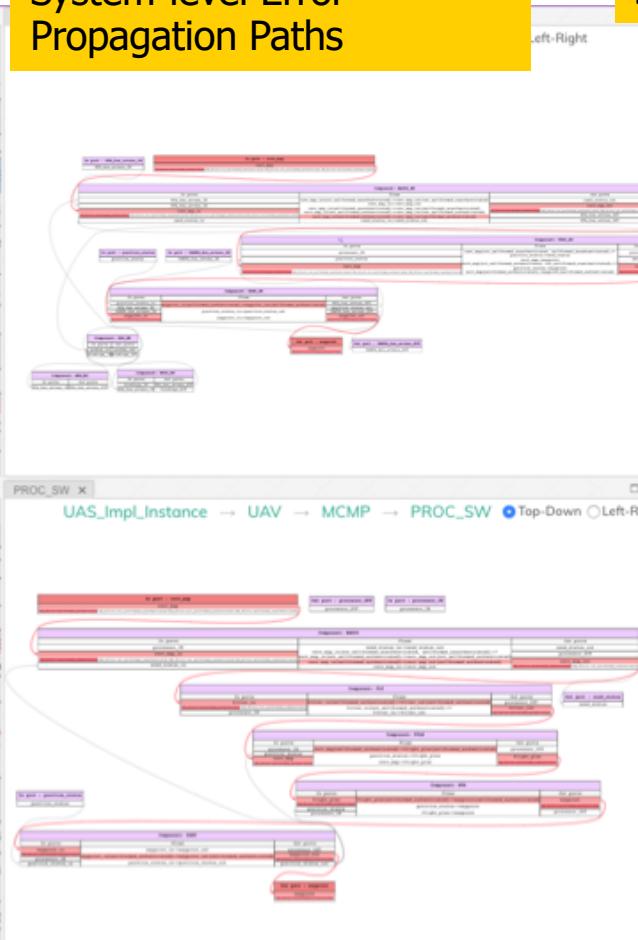


Example Visualization of AADL EMv2 Error Propagation (Hazard Analysis)

Details of intra-component Error Propagation



System-level Error Propagation Paths



Saved (replayable) queries

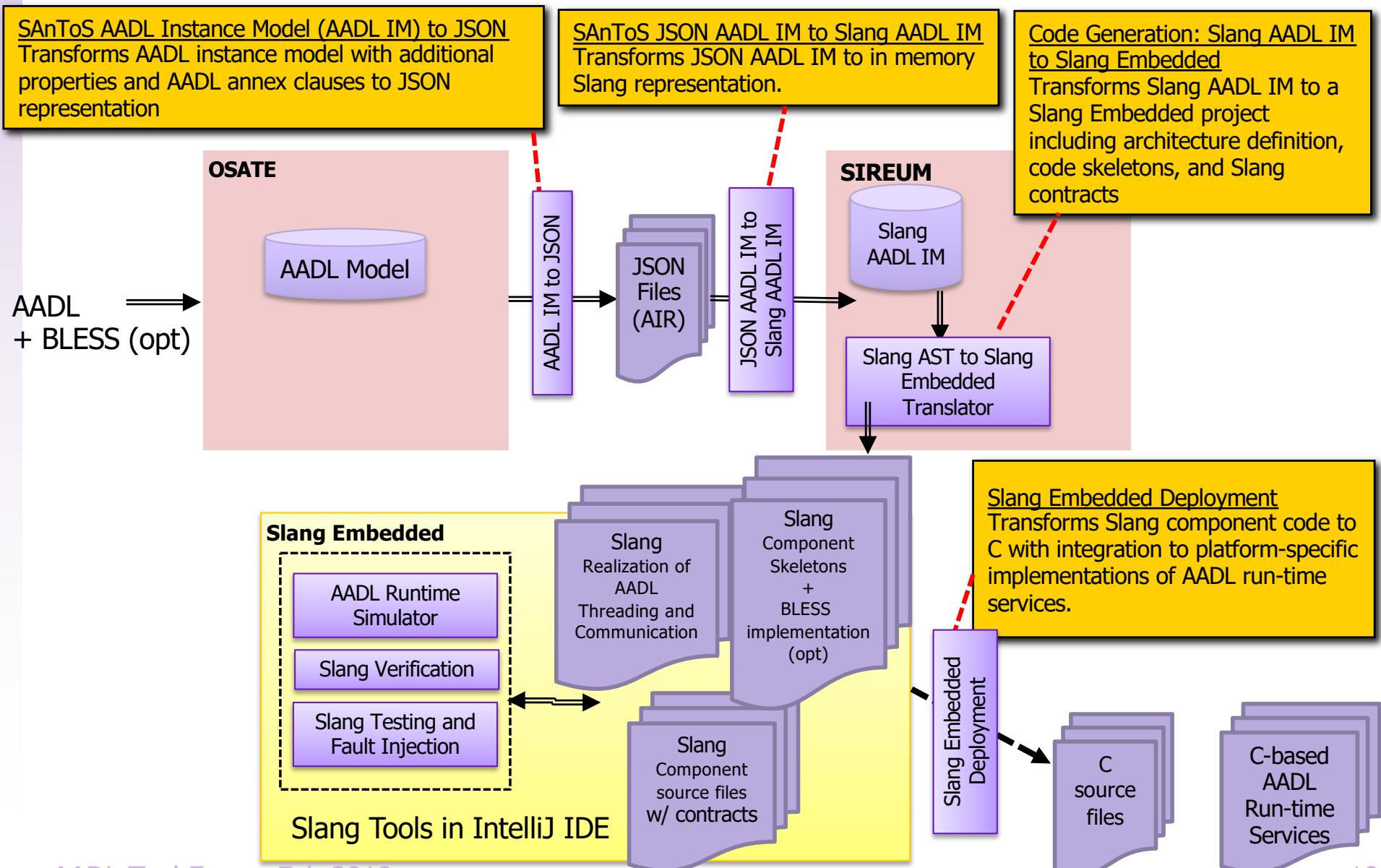
Table		
forward_GND	reach for	UAS_Impl
forward_GND_send_map	reach for	UAS_Impl
backward_GND_status	reach bac	UAS_Impl
GS_flight_controller	reach fro	UAS_Impl
GS_flight_controller_paths	reach pat	UAS_Impl
GS_flight_controller_refined_paths	reach refi	UAS_Impl
GS_flight_controller_refined_paths_without_flt	reach refi	UAS_Impl
only_wellformed_authorized	reach refi	UAS_Impl

```

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

```

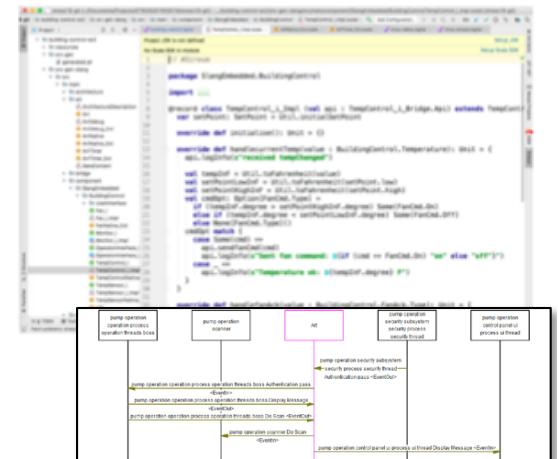
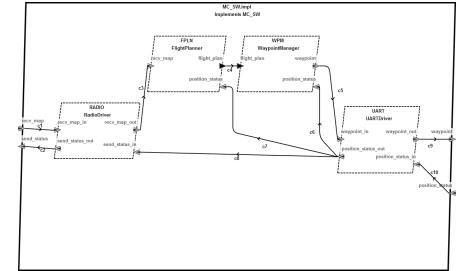
Code Generation Architecture



Code Generation

AADL Model-based Code Generation, Application Development, Simulation/Testing – with focus on separation kernels (e.g., SeL4)

- Code generation from AADL to C
- Code generation can be factored through Slang – a safety-critical subset of Scala – to provide automated source code verification and integration with JVM-based languages like Java, Scala
- Flexible backend can support a variety of middleware and platform targets
- Special focus on applications on separation kernels (e.g., SeL4 – being used on DARPA CASE)
 - Unix / Linux
 - Minix 3 plus extensions
 - SeL4
 - In progress: FreeRTOS, Genode
- Architecture permits easy incorporation of different communication middleware, e.g.,
 - In progress: DDS, CAN



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

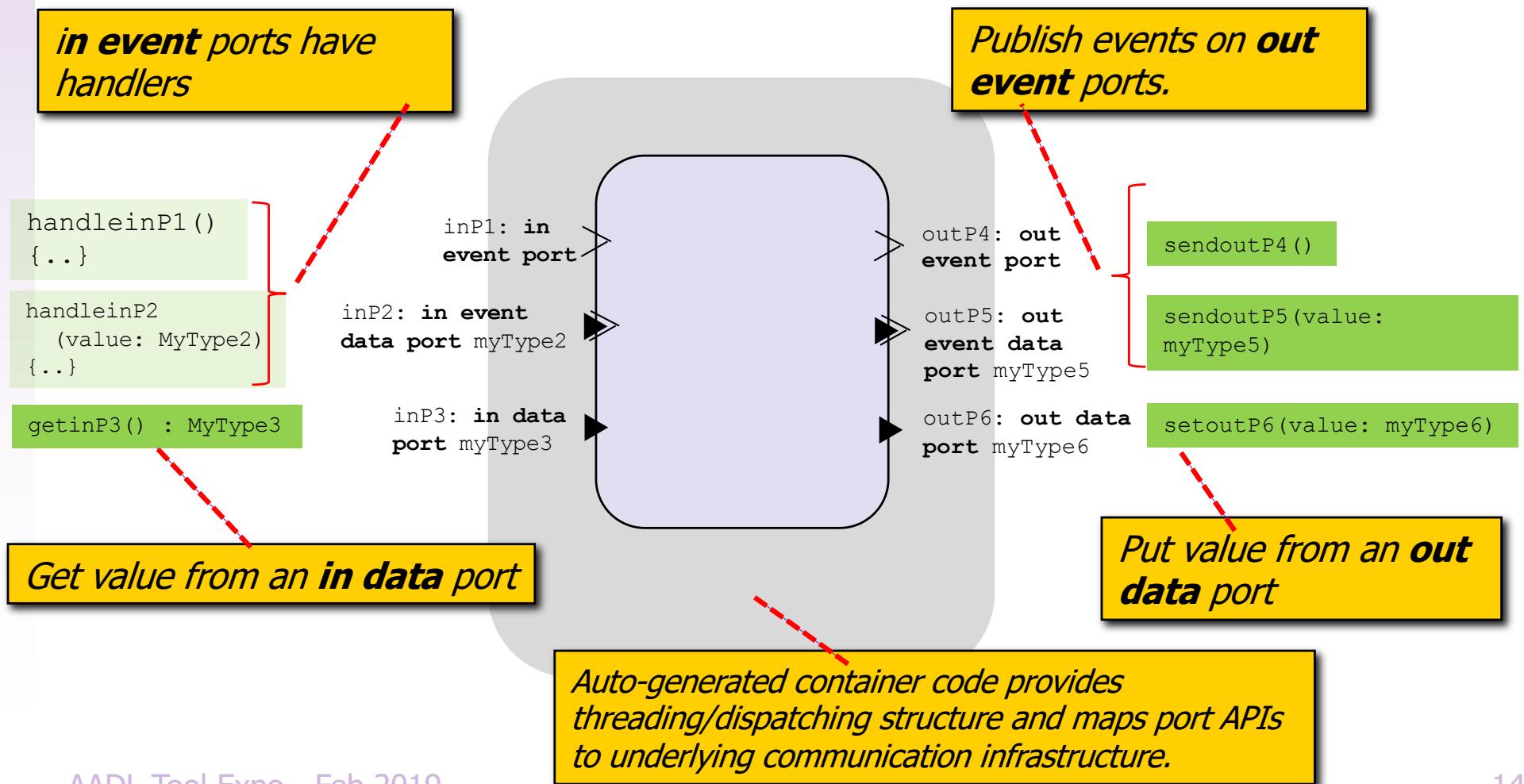
- Slang -- A verifiable subset of a modern programming language — Scala
 - **imperative OO & FP**: generics, pattern matching, higher-order functions, etc.
 - **benefits**: existing Java ecosystems and talent pools, available (customizable) industrial tool support, including compiler toolchain & IDEs
 - ... 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

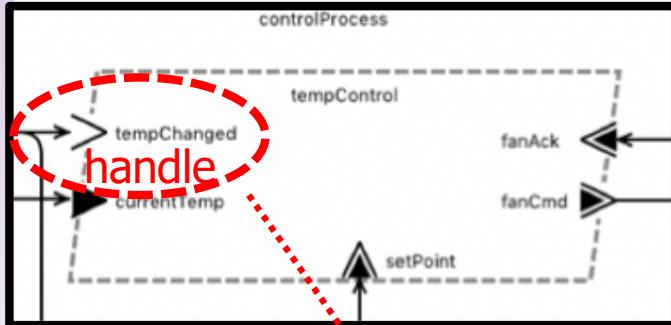
- **C Standard:** C99, **Compilers:** CompCert (proven correct C compiler), clang, gcc
- **OS/platforms:** macOS, Linux, Windows, and others (opportunity-based)
- **Memory models:** static alloc. (done); ref-counting & full tracing-GC (*future*)
- **Platform Backends**
 - Conventional C applications 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

Models as Program Feature Directives

Selection of **port types (in/out event/data)** “programs” the structure of the business logic of the component as well as the infrastructure code linking the communication and scheduler to the business logic.



Component Implementations in Slang

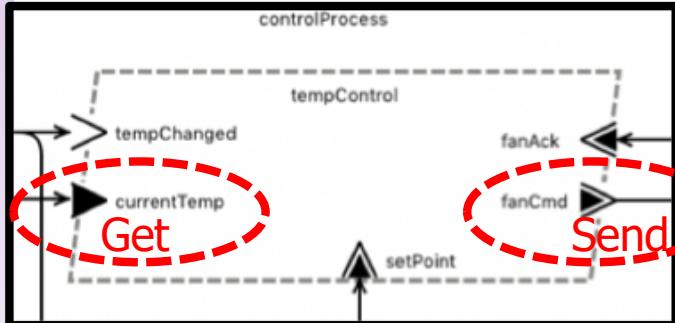


...Slang is used to implement component business logic (corresponding to event handlers for incoming interface events)

```
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 (cmd == FanCmd.On) "on" else "off"}")
        case _ =>
            api.logInfo( msg = s"Temperature ok: ${tempInF.degree} F")
    }
}
```

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 "off"}")
        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)

Component Implementations in Slang Compiled from BLESS/BA

BLESS/BA transition systems are compiled down to a straightforward representation in Slang

```
1  var currentState : CompleteState.Type = CompleteState.start
2  var last_patient_bolus : Art.Time = z"0"
3
4  def Compute_EntryPoint(): Unit = {
5    currentState match {
6      case CompleteState.start =>
7        case CompleteState.run =>
8          if (api.getPatient_Button_Request().nonEmpty) { do_button() }
9          ...
10     }
11   }
12
13  def do_button(): Unit = {
14    executionState_check_last_bolus_time()
15  }
16
17  def executionState_check_last_bolus_time (): Unit = {
18    val t1 = Conversions.toArtTime(api.getMinimum_Time_Between_Bolus())
19    if (Art.time()-t1 > last_patient_bolus) { do_nottoosoon() }
20    else if (Art.time()-t1 <= last_patient_bolus) { do_toosoon() }
21  }
22
23  def do_nottoosoon(): Unit = {
24    api.sendPatient_Request_Not_Too_Soon()
25    last_patient_bolus = Art.time()
26    currentState = CompleteState.run
27  }
28
29  def do_toosoon(): Unit = {
30    api.sendPatient_Request_Too_Soon()
31    currentState = CompleteState.run
32 }
```

Variable holding the current complete state (set of complete states represented as Slang enum).

Compute_EntryPoint (AADL RT) switches to current complete state then calls a method to that selects and executes transitions from complete state.

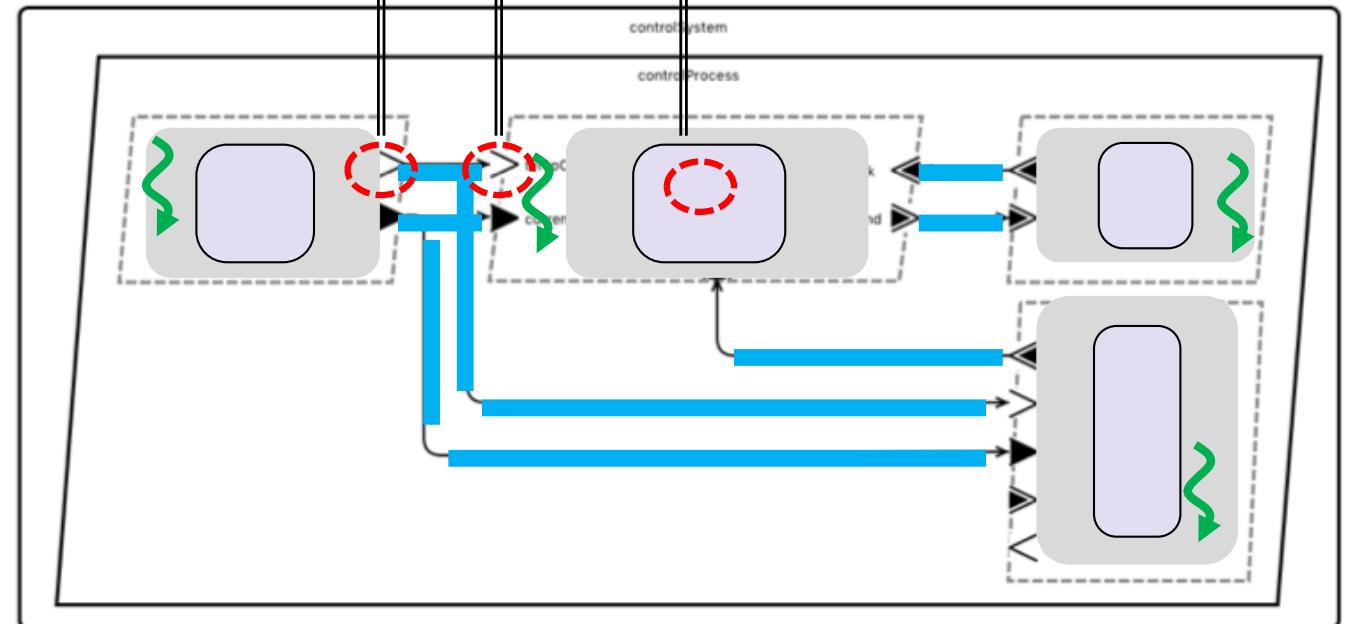
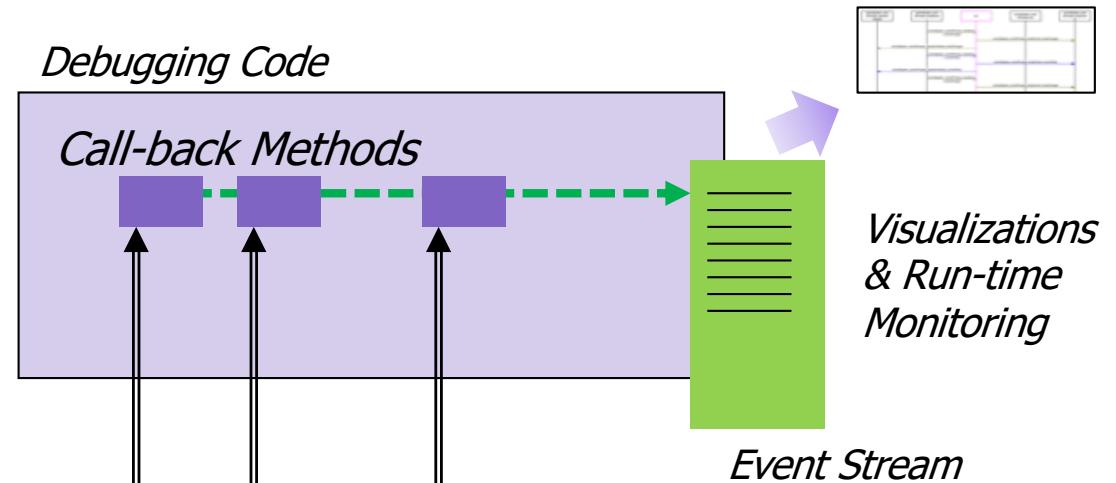
Check data port as part of dispatch condition.

Execute active transition for check_last_bolus_time state.

Each transition, represented as a method, calls port functions (AADL RT behind the scenes) to send messages

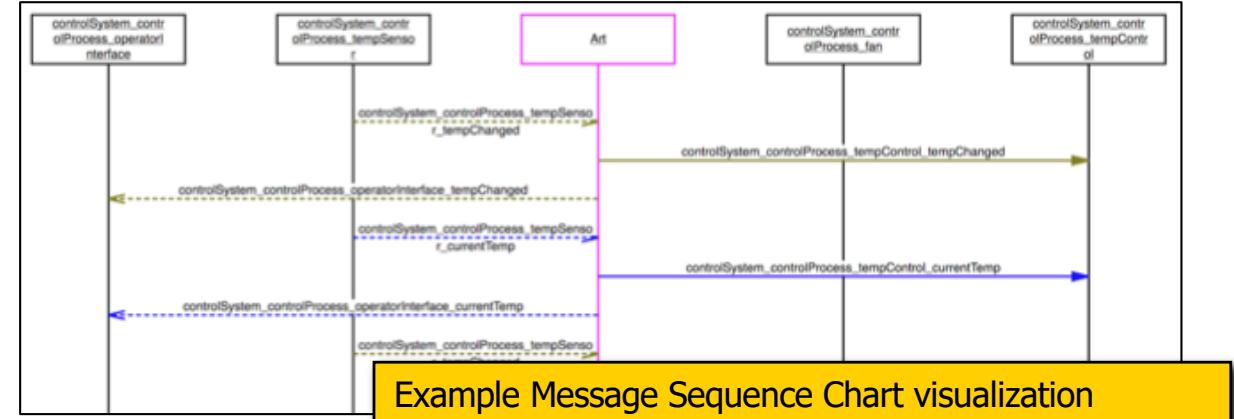
Slang Runtime Debugging Framework

The Slang 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 **component local variable changes**.

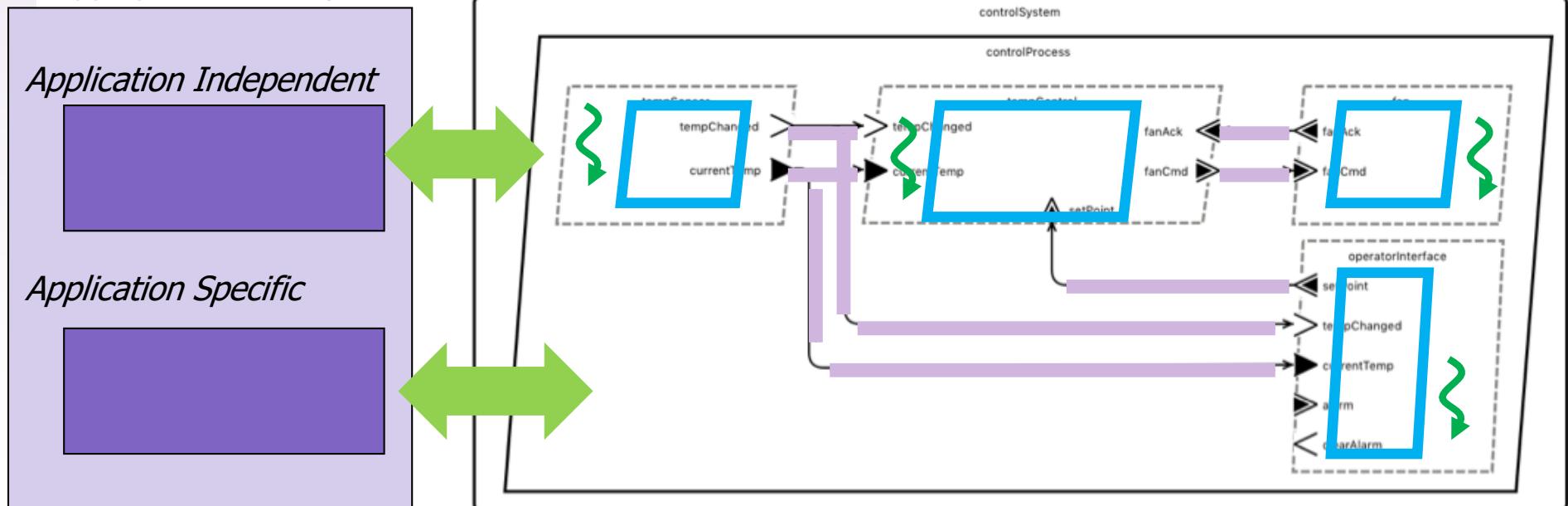


Slang Runtime Debugging Framework

By building a “watcher” for all send / receive actions on ports, and formatting info for a standardized Message Sequence Chart format, we can rapidly develop an MSC visualization of system as it is running.

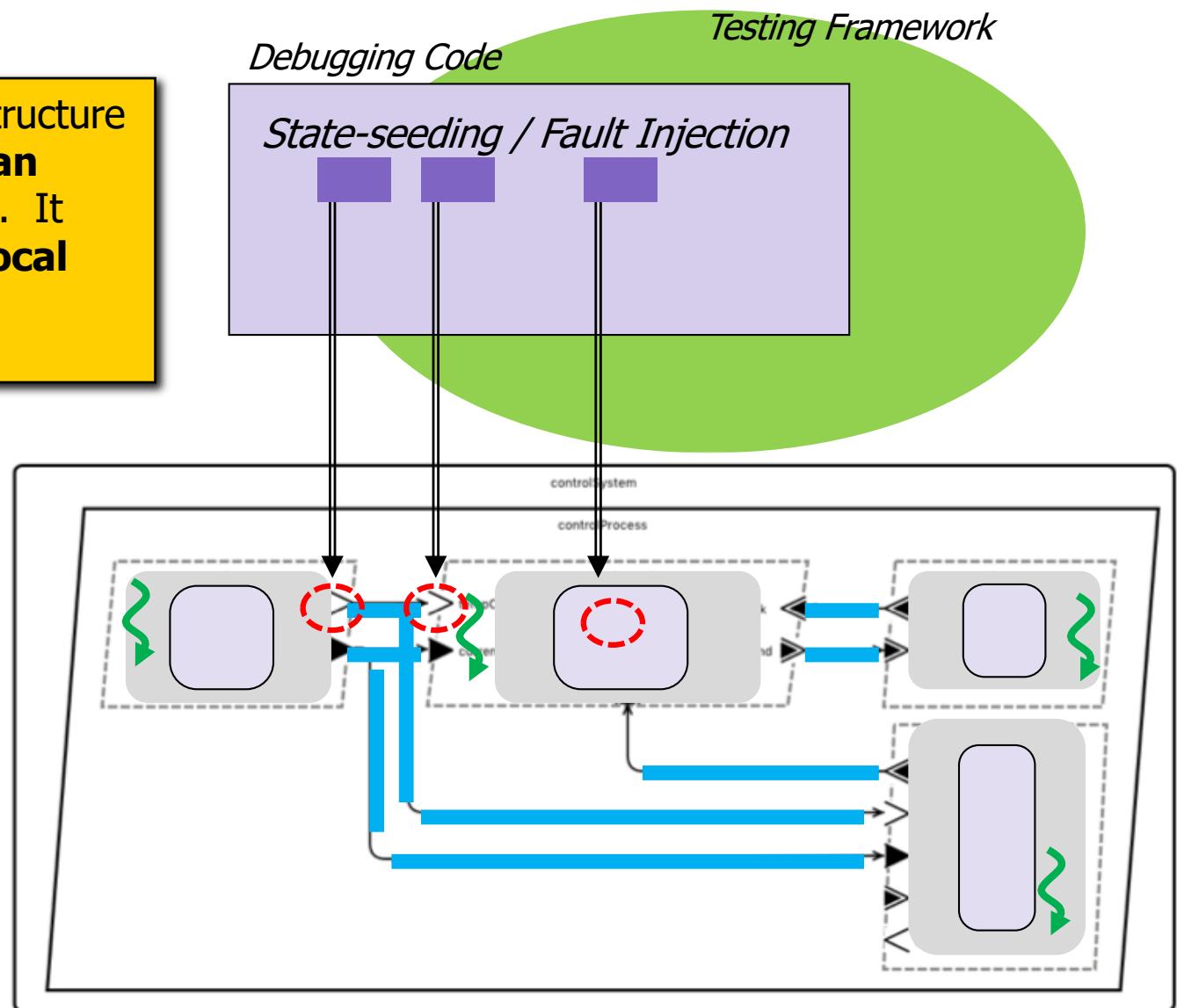


Debugging Functionality

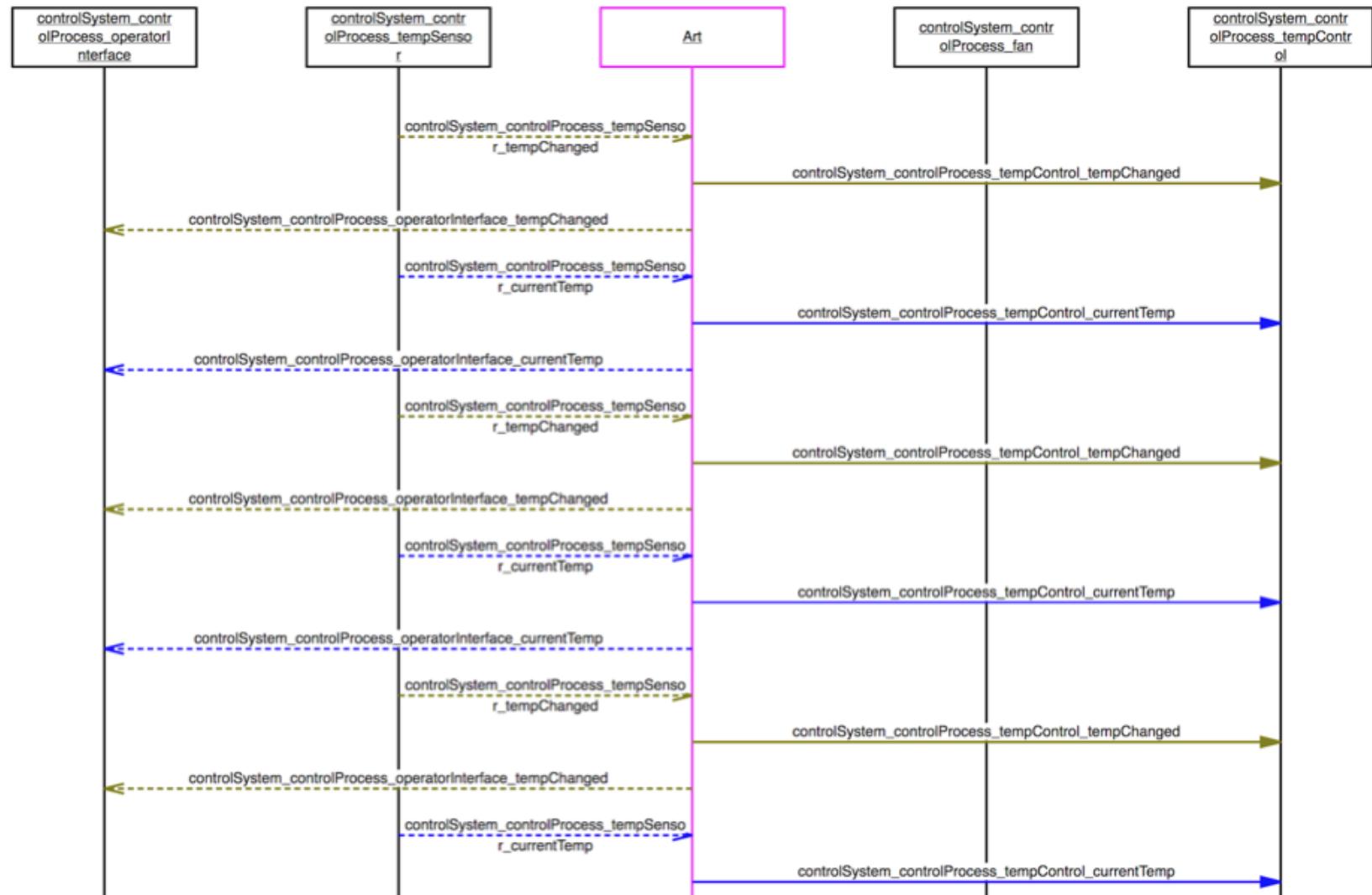


Slang Runtime Debugging Framework

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



Example MSC Visualization

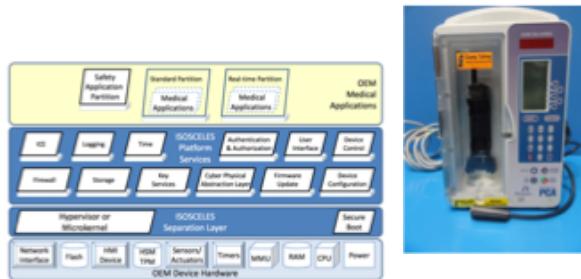


Conclusions

- Information flow and code generation capabilities working on “industrial strength” examples (DARPA and DHS research projects)
- Multiple on-going research projects to provide in enhancements
- Significant emphasis on...
 - targeting real-time systems on partitioning architectures
 - verification for high-assurance
- Related demos...
 - Adventium Labs
 - BLESS

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)

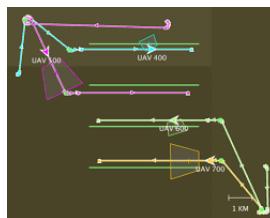


Containment labs for critical agriculture experiments

Code deployed using enhanced Minix 3 micro-kernel

- 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

UxAS – Unmanned (AFRL, DARPA)



Unmanned Systems Autonomy Services



Code deployed on machine-verified micro-kernel SEL4

Military Avionics

