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 Bureau 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

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

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

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

- 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.

"Before" (Federated)

"After" (Integrated, Cyber-resilient)
Deeply Integrate Models and Programming Across Multiple Levels of Abstraction

**System Modeling and Analysis (AADL)**

- AADL OSATE

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

**Deployment on Embedded/Distributed Platforms**

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

**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

**Source Code**, Simulation, Analysis, Verification

Semantic Consistency

Analysis and verification results moved up and down abstraction layers
Example Domains

**Medical Devices (US Dept of Homeland Security)**
- Targeting 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

**Building Controls (US Dept of Homeland Security)**
- Code deployed using Genode OS framework using Xen Hypervisor and SeL4 microkernel

**UxAS – Unmanned (AFRL, DARPA)**
- Code deployed using enhanced Minix 3 micro-kernel

**Containment labs for critical agriculture experiments**

**UxAS – Unmanned Systems Autonomy Services**

**NASA/JPL**
- Code deployed on machine-verified micro-kernel SEL4

AADL Tool Expo - Oct 2019
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

System Build

Platform configuration information

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 Component Infrastructure Code for Platform

Application Code

Application Code

Application Code

Application Code

Code gen for Application APIs

Code gen for Application APIs

Code gen for Application APIs

Code gen for Application APIs

Code gen for Component & Threading Infrastructure

Code gen for Communication Infrastructure

Code gen for Communication Infrastructure

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

Component Infrastructure in C, talking to SeL4 communication mechanisms

SeL4 Interpartition Communication in C

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

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

Platform-independent C code generation for AADL RT APIs

Code generation pathways for SeL4

AADL Tool Expo - Oct 2019
Use Case: Example HAMR instantiation for C-based development on Linux (e.g., DARPA CASE)

Component Infrastructure in C, talking to Linux inter-process communication

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

Linux inter-process communication in C

Platform independent C code generation for AADL RT APIs

Code generation pathways for Linux
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

...i.e.g., in C with platform infrastructure
HAMR Run-time Reference Implementation

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

- 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

Army SBIR "GUMBO" Adventium/KSU

AADL Tool Expo - Oct 2019
Component Implementations in Slang

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

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

  val tempInf = Util.toDouble(api.getCurrentTemp().get)
  val setPointLowInf = Util.toDouble(setPoint.low)
  val setPointHighInf = Util.toDouble(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: $cmd")
    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

```scala
override def handleTempChanged(): Unit = {
  api.logInfo(msg = "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 = "Sent fan command: \${if (cmd == FanCmd.On) "on" else "off"}
    case _ =>
      api.logInfo(msg = "Temperature ok: \${tempInF.degree} F")
  }
```

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

Reading a value from the currentTemp data port (behind the scenes mapped to generic AADL RT service GetValue)
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 an application component local variable changes.

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
Example Event Stream Filtering

Inspector/Injector Framework

Filtered event stream for temp sensor fault injection path

Get identifiers of ports of interest

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

Define stream start / end points

I’d like to visualize events on the temp sensor fault mitigation 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"
}
## Event Filtering

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

**Event stream**

<table>
<thead>
<tr>
<th>Filter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>manual-periodic-only</td>
<td></td>
<td>40 s 281 ms</td>
</tr>
<tr>
<td>manual-sporadic-only</td>
<td></td>
<td>41 s 281 ms</td>
</tr>
<tr>
<td>manual-start-only-live-5-sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>manual-stop-only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>out-port-only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>periodic-only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sporadic-only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>temp-sensor-alarm</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example:**

```
Port: BuildingControlDemo_1_Instance_tcp_tempControl (1) Sporadic(1000)
Time: 40 s 281 ms
Data: Temperature_Payload(Temperature(87.33246f, Fahrenheit))
```
Auto-generated Sequence Chart Visualization

Clicking on button here automatically moves from text-based view to sequence chart view.
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**.
The KSU Awas tool builds scalable interactive visualizations of AADL information flows and error propagations.

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

Results from DoD Phase II SBIR with Adventium Labs
Internal dependency graphs upon which analysis is performed are built from architecture connections and intra-component flows as well as EMv2 annotations.
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 “forward propagation” option.

Immediately see results of across different subsystems.
Example Representation of AADL EMv2 Error Propagation (Hazard Analysis)

<table>
<thead>
<tr>
<th>Component: MCMF</th>
<th>In ports</th>
<th>Flows</th>
<th>Out ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>UARTA bus access IN</td>
<td>recv_map{wellformed}</td>
<td>waypoint{wellformed}</td>
<td>RFA bus access OUT</td>
</tr>
<tr>
<td>RFA bus access IN</td>
<td>recv_map{not_wellformed}</td>
<td>waypoint</td>
<td>send_status</td>
</tr>
<tr>
<td>recv_map</td>
<td>position_status</td>
<td>waypoint</td>
<td>UAS_Errors.wellformed</td>
</tr>
<tr>
<td>position_status</td>
<td>send_status</td>
<td>UARTA bus access OUT</td>
<td></td>
</tr>
</tbody>
</table>

Path of EMv2 error token propagation.

Highlighting of error tokens relevant to given query.

<table>
<thead>
<tr>
<th>Component: SBUS</th>
<th>In ports</th>
<th>Out ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>bindings IN</td>
<td>ACCESS OUT</td>
<td>send_status</td>
</tr>
<tr>
<td>ACCESS IN</td>
<td>bindings OUT</td>
<td></td>
</tr>
</tbody>
</table>

Visualization of EMv2 error token propagation rules.

<table>
<thead>
<tr>
<th>Component: FCTL</th>
<th>In ports</th>
<th>Flows</th>
<th>Out ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>waypoint</td>
<td>waypoint</td>
<td><em>-&gt;</em></td>
<td>position_status</td>
</tr>
<tr>
<td>UAS_Errors.wellformed</td>
<td>waypoint{wellformed}</td>
<td><em>-&gt;</em></td>
<td>UARTA bus access OUT</td>
</tr>
<tr>
<td>UARTA bus access IN</td>
<td>*-&gt;position_status</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

AADL Tool Expo - Oct 2019
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