HAMR -
High-Assurance Modeling and Rapid Engineering
for Embedded Systems Using AADL

HCSS 2021 – May 6, 2021

Kansas State University

John Hatcliff
Robby
Jason Belt

Adventium Labs

Todd Carpenter
John Shackleton

Collins Aerospace
Data61

This material is based on research sponsored in part by DARPA, US Army, AFRL, and the Software Engineering Institute

DISCLAIMER: The views and conclusions contained in this presentation are those of the author and should not be interpreted as representing the official policies, either express or implied, of any agency or department of the U.S. Government, Kansas State University, Adventium Labs, Collins Aerospace, or Data61.
What is HAMR?

HAMR is a model-driven development tool-chain for high-assurance embedded systems.

Modeling, analysis, and verification in the AADL modeling language.

Component development and verification in multiple languages:
- C
- Slang (safety-critical subset of Scala with a contract verification framework)
- CakeML (ML-variant with verified compiler)

Deployments aligned with AADL run-time on multiple platforms.

Leveraging analysis from AADL community.
DARPA CASE Approach

HAMR is being developed by Kansas State and Adventium Labs on a team led by Collins Aerospace (Darren Cofer) that includes Data61 and University of Kansas

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

HAMR - Hatcliff -- Kansas State

seL4 verified micro-kernel technology is a core technology

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

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

Control non-interference by allocating components to different partitions in microkernel
HAMR for seL4 Development

What does HAMR provide seL4 application developers?

A full systems engineering environment based on an standardized modeling language (AADL) with many accompanying analysis and verification tools + integration with industrial workflows

HAMR - Hatcliff -- Kansas State
HAMR for seL4 Development

What does HAMR provide seL4 developers?

**A full systems engineering environment** based on an standardized modeling language (AADL) with many accompanying analysis and verification tools + integration with industrial workflows.

**HAMR**

- **System Engineering**
  - AADL OSATE IDE

- **CAMKES Specification**
  - Semantics-preserving translation

- **HAMR**
  - Ensuring AADL modeling, analysis, and verification results carry over to implementation

**HAMR Goal**

Framework for building **verified applications** on top of seL4 verified infrastructure (DARPA, US Army, AFRL SBIRs w/ Adventium)

**HAMR - Hatcliff -- Kansas State**
HAMR for seL4 Development

What does HAMR provide seL4 developers?

**A full systems engineering environment** based on an standardized modeling language (AADL) with many accompanying analysis and verification tools + integration with industrial workflows

- AADL OSATE IDE
- OSATE + HAMR Capabilities
- Software/Hardware/Middleware Modeling
- Hazard Analysis
- Information Flow Analysis
- Timing / Scheduleability
- Component Contracts + Verification
- Unit Testing
- Simulation and Execution Visualization

**HAMR**

- Component application logic programmed in **C** or **Slang** (Scala subset and compilable to **C**)

**CAMKES Specification**

**Linux Deployment**

**JVM**

(Java Virtual Machine)
Example HAMR Multiple-Platform Workflow on DARPA CASE

Rapid prototyping / agile development progression to seL4 deployment

- Initial system model
- System message types designed
- Integration planned
- Component unit testing

**Platform:** JVM

**Slang** mock-up of component functions

**HAMR JVM Simulation and Visualization**
HAMR JVM Event Stream Filtering

HAMR JVM platform enables very flexible filtering and visualization of inter-component communication, with the ability to filter on different categories of messages, ports, components, etc.

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

Event stream
HAMR JVM Execution Visualizations

HAMR JVM provides ability to visualize system execution – below, dynamically generated/updated message sequence charts (filterable) of inter-component communication.
Example HAMR Multiple-Platform Workflow on DARPA CASE

Rapid prototyping / agile development progression to seL4 deployment

Initial system model

- Platform: JVM
  - Initial system model
  - System message types designed
  - Integration planned
  - Component unit testing

Slang mock-up of component functions

- Platform: Linux
  - C coding of component functions
    (hand-written or translated from Slang)

- Platform: seL4 + Qemu
  - Design of domain schedule
  - Integration of VMs
  - Enhanced system testing

- Platform: seL4 + board
  - Integration of drivers
  - Final system testing
  - Penetration testing

HAMR JVM Simulation and Visualization

Testing of C components
Mock up of VM functions
Initial System Testing
Notions of Composition

**Horizontal Composition**

...composing components via AADL port connections at the application layer

**Vertical Composition**

...composing multiple layers of abstractions for threading and communication (refinements)

...ensuring consistency in threading and communication structure to ensure correctness and enable horizontal composition at lower levels
AADL Computational Model

Developer configures computational structure

AADL Thread Property Options
- Periodic
- Sporadic
- Hybrid

AADL Port & Connection Property Options
- Event Data
- Event Data
- buffered notifications
- shared data cells
- (or data distribution service)
- buffered messages
- (message passing middleware)
- + QoS, buffer sizes, latencies, etc

Selected thread pattern

Implied API Pattern for application code to access port communication, etc.

Selected communication pattern
HAMR Code Generation

System Build

Platform configuration information

Auto-generated Component Infrastructure Code for Platform

Auto-generated Component Infrastructure Code for Platform

TempControlProcess.i

tempSensor

Changed

tempControl

fanCmd

fanAck

Code gen for Application APIs

Code gen for Application Code Development

Code gen for Application Code

Code gen for Component & Threading Infrastructure

Code gen for Communication Infrastructure

Auto-Generated Run-Time Communication Infrastructure Code for Platform

Auto-Generated Run-Time Communication Infrastructure Code for Platform

Auto-Generated Run-Time Communication Infrastructure Code for Platform

Auto-Generated Run-Time Communication Infrastructure Code for Platform

HAMR - Hatcliff -- Kansas State
On each dispatch, AADL threads follow a well-known **input-compute-output** pattern for real-time tasks that aid analysis and verification...

(1) Receive inputs

(2) Compute, Run to Completion

(3) Release outputs

AADL Component  Application Memory Boundary

**Input Ports**
- User Code
- **Output Port Variables**

**Communication Infrastructure - Inputs**
- Reads

**Communication Infrastructure - Outputs**
- Writes

From AADL standard...

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

...For each port, HAMR generates an API for communicating over that port. These link to auto-generated implementations of port communication for the chosen platform.

**APIs for Port Communication**

- `bool api_get_currentTemp(Temp value);`
- `bool api_get_fanAck(FanAck_Type *value);`
- `bool api_get_setPoint(SetPoint value);`
- `void api_send_fanCmd(FanCmd_Type value);`
- `bool api_get_tempChanged();`

**Implementations for Port Communication APIs for target platform**

- **Linux**
- **seL4** Uses CAmkES + HAMR-generated “glue code”
Platform-Independent Thread Structure Generated from AADL Model - *Periodic*

`HAMR - Hatcliff -- Kansas State`
Platform-Independent Thread Structure
Generated from AADL Model - Periodic

```java
Unit timeTriggered() {
    DeclNew_Temperature(currTemp);

    // read current temp from hardware sensor
    senseTemperature(&currTemp);

    // take action if temperature has changed
    if(lastTemperature.degrees != currTemp.degrees) {
        lastTemperature = currTemp;
        api_send_currentTemp(&lastTemperature);
        api_send_tempChanged();
    }
}
```

**Periodic thread – skeleton filled with application logic**

- `tempSensor`:
  - `tempChanged`
  - `currentTemp`

- `notification event`
- `data`

- `auto-generated`

- **Application Code Development**

- **Application Code**

- **Put new temp value on output data port**

- **Notify consumers that temperature changed**
Platform-Independent Thread Structure Generated from AADL Model - *Sporadic*

### AADL Model

- **Implied Semantics**
- **Application Code**
  - **Sporadic thread** – skeleton for application logic – **event handlers**

```c
void TempControl_tempChanged(void) {
    ...
}
```
Platform-Independent Thread Structure
Generated from AADL Model - *Sporadic*

**AADL Model Implied Semantics**

**Sporadic thread – skeleton for application logic – event handlers**

```c
void tempChanged(void) {
    DeclTemp(currTemp); // macro to allocate Temp in stack
    if (api_get_currentTemp(&currTemp)){
        struct Temp currTempInF = convertToF(&currTemp);
        if (currTempInF.degrees > setPoint.high.degrees) {
            api_send_fanCmd(FanCmd_On);
        } else
        if (currTempInF.degrees < setPoint.low.degrees) {
            api_send_fanCmd(FanCmd_Off);
        }
    }
}
```

**Application Code Development**
Platform-Independent Thread Structure
Generated from AADL Model - *Sporadic*

void tempChanged(void) {
  DeclTemp(currTemp); // macro to allocate Temp

  if (api_get_currentTemp(&currTemp)) {
    struct Temp currTempInF = convertToF(&currTemp);

    if (currTempInF.degrees > setPoint.high.degrees) {
      api_send_fanCmd(FanCmd_On);
    } else
    if (currTempInF.degrees < setPoint.low.degrees) {
      api_send_fanCmd(FanCmd_Off);
    }
  }
}
Platform-Independent Thread Structure Generated from AADL Model - *Sporadic*

...HAMR also generates handlers for the remaining input event data ports

```c
void handlesetPoint(SetPoint value) {
    ...
}

Sporadic thread – setPoint– event handler

void handlefanAck(FanAck_Type value) {
    ...
}

Sporadic thread – fanAck– event handler
```
HAMR Code Generation
seL4 Platform

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

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

Consume system partitioning using Data61 CAmkES

Partition specified as a CAmkES Component

Communication specified using CAmkES Connectors
HAMR Code Generation
seL4 Platform

HAMR utilizes the Data61 CAmkES framework to create the final build for deployment.
CAmkES seL4 Configuration

AADL and Platform Independent Application Code

For each AADL thread component create an seL4 partition

component TempControl {
}

component Fan {
}

seL4 Platform Configuration Using CAmkES (auto-generated by HAMR)
CAmkES seL4 Configuration

AADL and Platform Independent Application Code

```plaintext
component TempControl {
    emits ReceiveEvent
    fanCmd_notification;
    dataport DataContent
    fanCmd_queue;
    ...
}
```

```plaintext
component Fan {
    consumes ReceiveEvent
    fanCmd_notification;
    dataport DataContent
    fanCmd_queue;
    ...
}
```

AADL Event Data port is represented using a CAmkES **notification** + **dataport** – introducing finer granularity as we move to platform.
CAAkES seL4 Configuration

AADL and Platform Independent Application Code

```plaintext
component TempControl {  
  emits ReceiveEvent fanCmd_notification;  
  dataport DataContent fanCmd_queue;  
  ...
}

component Fan {  
  consumes ReceiveEvent fanCmd_notification;  
  dataport DataContent fanCmd_queue;  
  ...
}

composition {  
  component TempControl tempControl;  
  component Fan fan;  
  connection seL4Notification conn4(  
    from tempControl.fanCmd_notification,  
    to fan.fan_notification);  
  connection seL4SharedData conn5(  
    from tempControl.fanCmd_queue,  
    to fan.fanCmd_queue);  
  ...
}
```

CAAmkES assembly specifies system topology, including allowed communication between seL4 partitions.

seL4 Platform Configuration Using CAAkES (auto-generated by HAMR)
CAmkES seL4 Configuration

AADL and Platform Independent Application Code

```plaintext
component TempControl {
    emits ReceiveEvent
    dataport DataContent; tempControl.fanCmd_queue
}

component Fan {
    consumes ReceiveEvent
    dataport DataContent; fan.fanCmd_queue
}
```

seL4 Platform Configuration Using CAmkES (auto-generated by HAMR)

```plaintext
assembly {
    ... 
}
configuration {
    tempControl.fanCmdqueue_access = "W";
    fan.fanCmd_queue_access = "R";
    ... 
}
```

CAmkES **configuration** specifies seL4 capabilities for partition interaction.

Ensures that info flow implied by AADL model is achieved in deployed system using the formally verified seL4 info flow controls.
HAMR generates adapter code that realizes the AADL port queuing and thread dispatch semantics in terms of CAmkES/seL4 primitives.

HAMR inserts AADL-compliant component application code into CAmkES/seL4 partition.

Note: HAMR also includes an option for only generating the CAmkES configuration from the AADL model -- leaving the developer to do what they want with all the internals of the CAmkES component (i.e., infrastructure code for AADL semantics is not included).
VM Insertion

HAMR provides AADL modeling and code generation to automatically insert Linux virtual machines in to CAmkES/seL4 partitions (e.g., to host legacy or non-AADL-aligned code).

Guided by AADL model directives, HAMR can generate VM instances for an CAmkES/seL4 partition using pre-defined VM templates.

While some manual configuration must be done, HAMR provides various forms of automated support, e.g., generating Linux device table mappings to seL4 ports.

seL4 as configured by CAmkES

Legacy code
Sandboxed untrusted code, etc.

VM (Linux)

Linux device table mappings to seL4 ports
As a step towards the eventual use of seL4 MCS extensions, DARPA CASE is using the seL4 domain schedule to enforce temporal partitioning (see Todd Carpenter’s talk in Assured Systems II session).

```c
const dschedule_t ksDomSchedule[] = {
    { .domain = 0, .length = 100 }, // all other threads
    { .domain = 2, .length = 5 },   // tempSensor
    { .domain = 3, .length = 5 },   // tempControl
    { .domain = 4, .length = 5 },   // fan
    { .domain = 0, .length = 380 }, // pad rest of period
};
const word_t ksDomScheduleLength =
    sizeof(ksDomSchedule) / sizeof(dschedule_t);
```
Analyses and Verification enabled by planning for compositionality...
AADL Analysis Example: Information Flow

The KSU Awas tool ([https://awas.sireum.org](https://awas.sireum.org)) generates scalable interactive visualizations of AADL information flows and model-based hazard analysis results.

Information flow graphs can be dynamically browsed and queried with path logic.
Intra-Component Information Flow

Awas component visualizations focus developer attention on information flow aspects (other aspects of AADL specs are elided).
Internal dependency graphs upon which analysis is performed are built from architecture connections and intra-component AADL flow annotations as well as AADL EM annotations.

Inter-component info flow correctness is part of platform assurance (done once).

Intra-component info flow correctness is part of application assurance.

Platform assurance + Application assurance => System assurance.

Internal dependency graph (algorithms work on this).

Browser-based (HTML5).

Interactions and rendered results.
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.
HAMR generates graphs of CAmkES/seL4 platform flows (relying on assurance of CAmkES and seL4) along traceability information showing the correspondence between the model level specifications and realization of flow controls in the deployed code.
Platform Information Flow Assurance Sketch

AADL-level information flow graph

SMT-based proof of graph isomorphism/embedding

seL4 Platform-level information flow (at granularity of CAmkES topology)

Note: Application flow correspondence for AADL/Slang is part of Adventium/KSU AFRL SBIR
Contracts for Compositional Reasoning

On-going work: supporting compositional verification in AADL and HAMR-generated applications...

Existing AADL contract languages

- AGREE -- Assume-Guarantee REasoning Environment (used on DARPA CASE)
- BLESS – Behavior Language for Embedded Systems with Software
Contracts for Compositional Reasoning

On-going work: supporting compositional verification in AADL and HAMR-generated applications...

Current Army and DARPA SBIR projects with Adventium Labs

AADL contracts automatically translated to Slang

Using Slang verification to prove that component implementations conform to their AADL behavioral specifications.
Slang Contracts and Automated Verification via Symbolic Execution

Slang applications can be integrated with Scala and Java and executed on JVM or translated to C (generated C is compatible with verified CompCert compiler).

Verification Drill-down Controls

Slang Contract

Drill down display for verification conditions and SMT interaction

Application Code
Again, HAMR is leveraging the System Engineering emphasis of AADL – in particular, tools like Adventium’s FASTAR can process AADL model structure and timing annotations to perform schedulability analysis and automatically generate schedules.

**From Todd Carpenter’s “Time Enough for seL4” talk in Assured Systems II session.**
Related Work

HAMR enhances the HACMS Collins/U Minn/Data61 Trusted Build concept to provide...

- Completely new translation architecture with traceability mechanisms and support for eventual tool chain verification
- Alignment with AADL semantics
- Implementation of standardized AADL run-time services (key abstraction layer)
- Multiple platform support
- Multiple language support (adding Slang and CakeML)
- Automated insertion of virtual machines in seL4
- True one-way communication in seL4 (removing back channels)
- Traceability artifacts and information flow topology preservation proofs
- Temporal separation using seL4 domain scheduler

Other notable works on AADL code generation

- Ocarina – code generation for Ada and C (RT-POSIX threading, Xenomai, RTEMS, ARINC 653))
- RAMES – code generation for C (RT-Posix threading, nxtOSEK (LEGO Mindstorms), POK ARINC653-compliant.
Conclusion

**Sireum HAMR**

- High Assurance Model-based Rapid Engineering of Embedded Systems

- HAMR helps support DARPA CASE goals by adopting multiple design goals that emphasize compositionality
- ... leading to a rich model-based systems engineering framework that supports multiple deployment platforms and multiple implementation languages
- ...emphasizing assurance and rigorous development practices

**Resources on HAMR web site**

- Distribution available for Windows, Linux, and Mac (also virtualized) [hamr.sireum.org](http://hamr.sireum.org)
- Documentation, examples, and tutorial material
- Educational resources -- slides, recorded lectures, and guided exercises for HAMR Slang back end
Questions

Sireum HAMR
High Assurance Model-based Rapid Engineering of Embedded Systems

http://hamr.sireum.org