Mixed-Trust Scheduling in AADL

November 2021

Aaron Greenhouse

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Copyright

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM21-1002

Mixed-Trust Scheduling in AADL

Carnegie Mellon University Software Engineering Institute

Modeling Mixed-Trust Scheduling in AADL

- Modeling Mixed-Trust Processors
- Modeling Mixed-Trust Tasks

OSATE Mixed-Trust Scheduling Plugin

Modeling Mixed-Trust Processors

To execute a mixed-trust task, a processor must have

- A Guest OS
- A HyperVisor

The Guest OS and HyperVisor are best represented in AADL using **`virtual processor`** components:

- A virtual processor "[represents] a virtual machine such as the Java VM, a partition of a processor, or a scheduler" [*Model-Based Engineering with AADL*, Feiler & Gluch].

Guest OS and HyperVisor as Virtual Processors

A **`virtual processor`** compoment must be **bound** to a **`processor`** component.

- This is an AADL language requirement.

Two ways to declare binding:

- Use the **`Actual_Processor_Binding`** property on the virtual processors.
  - Usually done using a contained property association.
- Declare the virtual processors as subcomponents of the processor.

Example: Binding Virtual Processors

```
system implementation Example1.impl
  subcomponents
    P: processor;
    GuestOS: virtual processor;
    HyperVisor: virtual processor;

  properties
    Actual_Processor_Binding =>
      (reference(P)) applies to
        GuestOS, HyperVisor;
end Example1.impl;
```

Explicit binding to processor.

```
processor implementation MTP.i
  subcomponents
    GuestOS: virtual processor;
    HyperVisor: virtual processor;
end MixedTrustProcessor.i;
```

Subcomponents are *implicitly* bound to the containing processor.

```
system implementation Example2.impl
  subcomponents
    P: processor MTP.i;
end Example2.impl;
```

Declaring a Mixed-Trust Processor

A processor with bound virtual processors is not necessary meant for mixed-trust scheduling.

- Virtual processors are used to represent many things.

Must **declare** that the processor is **intended for mixed-trust scheduling**.

- Use a property association to do this.
  - Indicates which virtual processor is
    - The Guest OS.
    - The HyperVisor.

Mixed-Trust Processor Property

```
property set Mixed_Trust_Properties is

  Mixed_Trust_Bindings: type record (
    GuestOS: reference (virtual processor);
    HyperVisor: reference (virtual processor);
  );

  Mixed_Trust_Processor:
    Mixed_Trust_Properties::Mixed_Trust_Bindings
    applies to (processor);

  -- . . .
end Mixed_Trust_Properties;
```

Example: Mixed-Trust Processor Property

```
system implementation Example1.impl
  subcomponents
    P: processor;
    GuestOS: virtual processor;
    HyperVisor: virtual processor;

  properties
    Mixed_Trust_Properties::Mixed_Trust_Processor =>
      [GuestOS => reference(GuestOS);
       HyperVisor => reference(HyperVisor);]
    applies to P;

    Actual_Processor_Binding =>
      (reference(P)) applies to GuestOS, HyperVisor;
end Example1.impl;
```

Modeling Mixed-Trust Tasks — Pairing Threads

A mixed-trust task is fundamentally a pair of threads:

- One executes on the Guest OS.
- One executes on the HyperVisor.

Use standard AADL thread binding mechanism to assign threads to virtual processors:

- The **`Actual_Processor_Binding`** property (again).

But, need to know **which pair of threads** corresponds to a single mixed-trust task.

- Again, **declare** this using a property association.

Modeling Mixed-Trust Tasks — Timing

Each mixed-trust task has timing information:

- Period
  - Applies to the mixed-trust task, not just a single thread
    - Standard AADL `Period` property would need to be consistent across the thread pair.
- Deadline
  - Applies to the mixed-trust task, not just a single thread
    - Standard AADL `Period` property would need to be consistent across the thread pair.
- Execution times
  - Applies to the individual threads
    - Use the standard AADL `Compute_Execution_Time` property.

Mixed-Trust Tasks Property

```
property set Mixed_Trust_Properties is
  Mixed_Trust_Task: type record (
    Name: aadlstring;
    Period: Time;
    Deadline: Time;
    GuestTask: reference (Thread);
    HyperTask: reference (Thread);
    E: Time;
  );

  Mixed_Trust_Tasks:
    list of Mixed_Trust_Properties::Mixed_Trust_Task
    applies to (System);

  -- . . .
end Mixed_Trust_Properties;
```

Used in place of the **Period** and **Deadline** properties on the threads themselves.

Really an output of the scheduler. Ideally this would be automatically inserted into the instance model.

Example: Mixed-Trust Tasks Property (1)

```
process MTT
end MixedTrustTask;

process implementation MTT.basic
  subcomponents
    GuestThread: thread;
    HyperThread: thread;
end MTT.basic;

process implementation MTT.t1
  subcomponents
    GuestThread: thread {
      Compute_Execution_Time => 4ms .. 4ms;
    };
    HyperThread: thread {
      Compute_Execution_Time => 0ms .. 0ms;
    };
  end MixedTrustTask.t1;
```

```
process implementation MTT.t2
  subcomponents
    GuestThread: thread {
      Compute_Execution_Time => 2ms .. 2ms;
    };
    HyperThread: thread {
      Compute_Execution_Time => 3ms .. 3ms;
    };
end MixedTrustTask.t2;
```

Current approach **does not** require the two threads to be in the same process, but it is a convenient abstraction.

Example: Mixed-Trust Tasks Property (2)

```
system implementation Example1.impl
  subcomponents
    P: processor;
    GuestOS: virtual processor;
    HyperVisor: virtual processor;
    MTT1: process MTT.task1;
    MTT2: process MTT.task2;
  properties
    Actual_Processor_Binding => (reference(P))
    applies to GuestOS, HyperVisor;

    Mixed_Trust_Properties::Mixed_Trust_Processor =>
      [GuestOS => reference(GuestOS);
       HyperVisor => reference(HyperVisor);]
    applies to P;

    Actual_Processor_Binding => (reference(GuestOS))
    applies to MTT1.GuestThread, MTT2.GuestThread;

    Actual_Processor_Binding => (reference(HyperVisor))
    applies to MTT1.HyperThread, MTT2.HyperThread;

    Mixed_Trust_Properties::Mixed_Trust_Tasks =>
      ([Name => "MT 1";
        Period => 8 ms;
        Deadline => 8 ms;
        GuestTask => reference(MTT1.GuestThread);
        HyperTask => reference(MTT1.HyperThread);],

       [Name => "MT 2";
        Period => 14 ms;
        Deadline => 14 ms;
        GuestTask => reference(MTT2.GuestThread);
        HyperTask => reference(MTT2.HyperThread);]);
end Example1.impl;
```

OSATE Analysis Plug-In

OSATE Plug-in builds an analysis around Dio's scheduling code.

- Based on the modeling approach just described.
- Performs consistency checking on the model.

Input is an AADL instance model.

- Checks each system operation mode for schedualability.
- Outputs results to a CSV file.
  - Schedulable?
  - E values.

Consistency Checks

The analysis checks consistency based on

- Model structure.
- Mixed-trust property associations.

Reports problems using Eclipse error markers.

For example

- A **`Mixed_Trust_Processor`** property association must provide values for both the **`GuestOS`** and **`HyperVisor`** fields.
  - (AADL does not require this.)
- The virtual processors referenced by the **`Mixed_Trust_Processor`** fields must be bound to the associated processor component.

More Consistency Checks

Checking **`Mixed_Trust_Tasks`** property associations:

- A **`GuestTask`** thread must be bound to a virtual processor identified as a **`GuestOS`**.
- A **`HyperTask`** thread must be bound to a virtual processor identified as a **`HyperVisor`**.
- The two virtual processors must be bound to the same processor.
  - Cannot split a task across *two different* mixed-trust domains.

There are more checks; this is to just give an idea of what is done.