Session 7: Modeling Software Runtime Characteristics

This is a hands-on exercise to become familiar with modeling runtime characteristics of a system.

### Goal of the Exercise:

* Reinforce the syntax and semantics of AADL software runtime components: process, thread, port connections, timing specifications and associated properties.
* Create a multi-threaded model of cruise control system.
* We will run the latency analysis for two flows

### Setting up the exercise:

We will begin with the completed model from Session 5. To ensure that everyone starts with the same model, we will import a completed model from the CD. The steps are:

1. Start OSATE and switch to a new workspace my\_session7\_workspace. This can be done by File -> Switch Workspace -> Other and typing in the path and the workspace name, e.g. my\_session7\_workspace
2. Import the model created in Session 5 that is on the DVD by:
   1. File -> Import-> General -> Existing projects into workspace
   2. The Import Wizard will appear. Set the Select Archive radio button, and use the browser button to browse to <prefix>/Session 5/session5\_solution.zip. <prefix> is the path to the session5\_solution.zip file to be imported. Click finish and the Session 5 project will appear in the AADL Navigator.

### Problem Scope:

This example focuses on the runtime specification and behavior of the cruise control components. A software application is usually decomposed into functional modules which are then assigned to multiple threads of execution. Having multiple threads and scheduling them appropriately allow higher CPU utilization. When control engineers develop a control algorithm, they generally assume their algorithm will run in one thread. Sometimes the decision is made by the software engineer to convert the control algorithms into a multi-threaded application.

### Problem Details:

Develop the solution using the text editor. We will then show how to create graphical diagrams from the finished model.

Assume that the cruise control has the following threads:

*ScanInputPorts*: Acquire data from sensors, apply digital filters where required.

*ComputeDesiredSpeed*: Control law, often a proportional-sum-difference control law

*ComputeThrottleSetting*: converts and scales the output to the throttle.

The thread communication is shown in Figure 1 below:

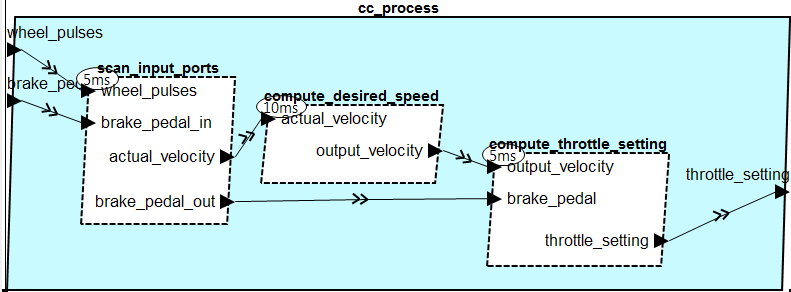


Figure 1: Cruise Control Process implementation containing three threads

### Suggested Solution Approach:

**Create Threads**

Create a new package file *CCThreads*, with a package name of *CCThreads*, by selecting New -> AADL model ->AADL package and naming the package *CCThreads*.

Within the package, define the following thread types with the appropriate flow paths (e.g. flow\_W\_T, flow\_B\_T, or both), and properties:

* Thread: *ScanInputPorts*, with I/O ports named as shown in the figure above.

Properti*es:* dispatch\_protocol = periodic, period = 5 ms, compute\_execution\_time = 3ms  
Specifically:

**thread** ScanInputPorts

**features**

wheel\_pulses: **in** **data** **port**;

brake\_pedal\_in: **in** **data** **port**;

actual\_velocity: **out** **data** **port**;

brake\_pedal\_out: **out** **data** **port**;

**flows**

flow\_W\_T: **flow** **path** wheel\_pulses -> actual\_velocity;

flow\_B\_T: **flow** **path** brake\_pedal\_in -> brake\_pedal\_out;

**properties**

Dispatch\_Protocol => Periodic;

Period => 5 Ms;

Compute\_Execution\_Time => 3 Ms .. 3 Ms;

**end** ScanInputPorts;

* Thread: *ComputeDesiredSpeed*, with I/O ports named as shown above,

Properties: dispatch protocol = periodic, period = 10 ms, compute\_execution\_time = 2 ms

Add the port features as shown above. The flow specification is:

**flows**

flow\_W\_T: **flow** **path** actual\_velocity -> output\_velocity;

* Thread: *ComputeThrottleSetting*, with I/O ports named as shown above

Properties: dispatch\_protocol = periodic, period = 5 ms, compute\_execution\_time = 3 ms

Add data ports as shown in the above picture. Use the following flow specifications:

**flows**

flow\_W\_T: **flow** **path** output\_velocity -> throttle\_setting;

flow\_B\_T: **flow** **path** brake\_pedal -> throttle\_setting;

Save the file.

**Create CruiseControlProcess**

So that an overall flow latency can be determined, you need to create a process in which to place the threads. Name the process CruiseControlProcess. Add the necessary in and out data ports as indicated in the above figure.

You will also need to specify the necessary flow paths:

* Flow from wheel to throttle: Flow\_W\_T
* Flow from brake to throttle: Flow\_B\_T

**Create CruiseControlProcess implementation**

Create CruiseControlProcess.impl by:

* Adding the thread subcomponents and name them: scan\_input\_ports, compute\_desired\_speed, and compute\_throttle\_setting.
* Add the connections naming them d1 through d6
* Add the flow path implementation for both flows, naming them: Flow\_W\_T, Flow\_B\_T.

Save the file.

This completes the thread modeling and creating the CruiseControlProcess and its associated implementation.

**Detailing Data Ports within the Existing Feature Groups**

To this point we have created a CruiseControlProcess implementation composed of three threads, and data ports as the external features, as indicated in Figure 1 above.

Since the thread inputs and outputs have been defined as having data ports, that means that the port group connections that we previously specified as empty, must now be detailed to include a data port name that corresponds to the connections from the wheel rotation and brake pedal sensors to the cruise control, and the cruise control connection to the throttle actuator and brake actuator.

**Detailing the Sensor Feature Group Types**

Open up the aadl -> packages -> Sensors.aadl file. You will notice that the Device WheelRotationSensor contains the feature group type named wrs\_cc\_type.

We will add a data port into the feature group with the appropriate direction.

**feature group** wrs\_cc

**features**

wheel\_pulses: **out data port**;

**end** wrs\_cc;

We do the same for the bp\_cc feature group type declaration.

**feature** **group** bp\_cc\_type

**features**

brake\_pedal: **out** **data** **port**;

**end** bp\_cc\_type;

**Detailing the Actuator Feature Group Type**

The port group associated with the throttle actuator must now be created and the corresponding device refined in a similar way to the wr\_cc port group. Open aadl->packages->Actuators and modify it in the following way by specifying the port group type and its inverse:

**feature** **group** ta\_cc\_type

**features**

throttle\_setting: **in** **data** **port**;

**end** ta\_cc\_type;

**Modifying the CruiseControl system in SoftwareApps**

Next we have to insert the AADL process CruiseControlProcess.impl as subcomponent. We do so by replacing the four subcomponents in CruiseControl.impl with a process subcomponent referencing CCThreads::CruiseControlProcess.impl. We will also the connections with appropriate connections from the enclosing system to the process. Finally, we update the two flow paths to go through the process subcomponent.

The new CruiseControl.impl will look like:

**system** **implementation** CruiseControl.impl

**subcomponents**

cc\_process: **process** CCThreads::CruiseControlProcess.impl;

**connections**

d1: **port** cc\_wrs.wheel\_pulses -> cc\_process.wheel\_pulses;

d2: **port** cc\_bp.brake\_pedal -> cc\_process.brake\_pedal;

d3: **port** cc\_process.throttle\_setting -> cc\_ta.throttle\_setting;

**flows**

Flow\_W\_T: **flow** **path** cc\_wrs.wheel\_pulses -> d1 -> cc\_process.Flow\_W\_T

-> d3 -> cc\_ta.throttle\_setting;

Flow\_B\_T: **flow** **path** cc\_bp.brake\_pedal -> d2 -> cc\_process.Flow\_B\_T

-> d3 -> cc\_ta.throttle\_setting;

**end** CruiseControl.impl;

Save the file.

**Graphical view of the new system**

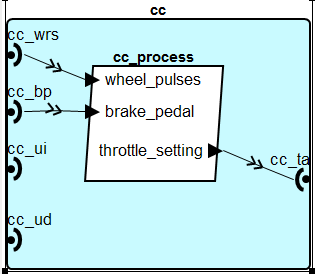
We view the process implementation in CCThreads by opening the file CCThreads.aadl in the Graphical Model Viewer.

We can also create the instance model for the MyCar implementation and open it in the Graphical Model Viewer.

We first see the top-level of the architecture, cruise control CC and its context components.

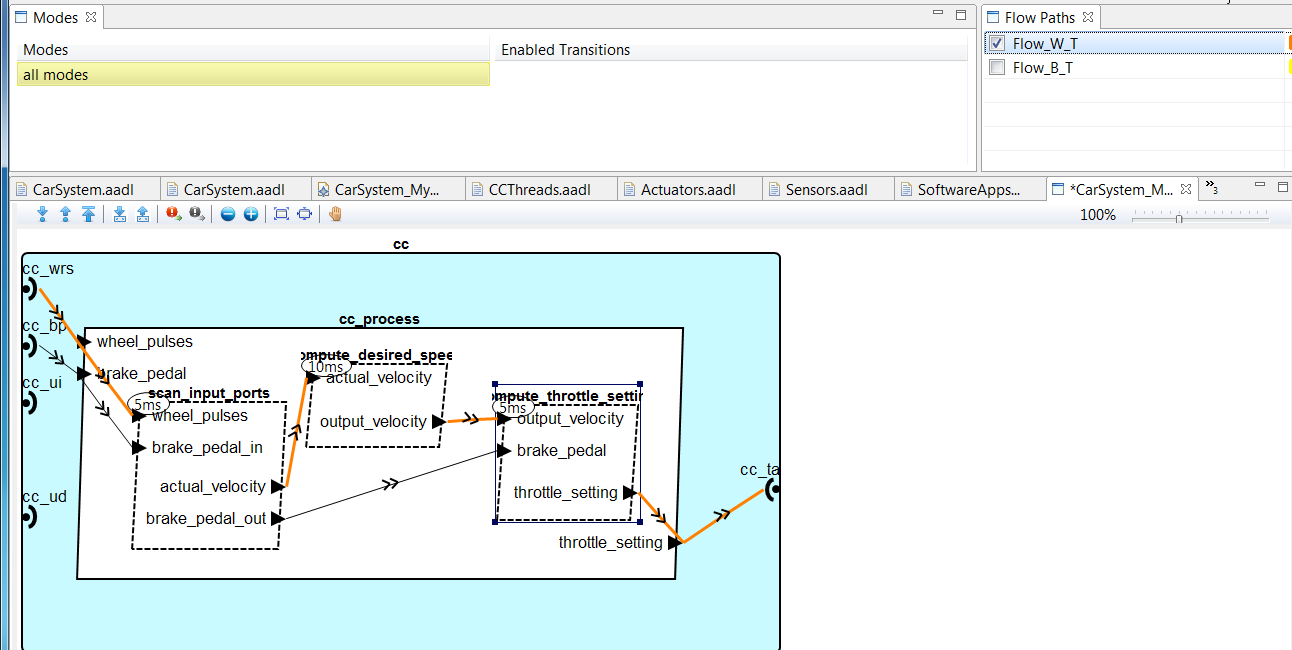
When we double click on CC the diagram opens into the diagram showing the context of CC, namely the cruise control process shown below.

When we double click on cc\_process the diagram opens into a diagram showing the threads inside the process.



Note that the Graphical Model Viewer allows you to see more than one level of the component hierarchy by clicking on  . The figure below shows the result when doing so inside the crusie control system CC.

Note also that the Graphical Model Viewer is able to highlight any flows we have specified. As we check one of the listed flow in the *Flow Paths* view the flow is highlighted in the diagram.

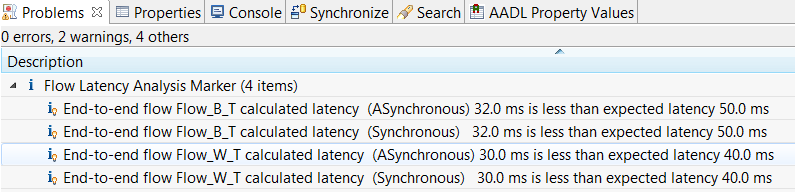


**Latency Analysis:**

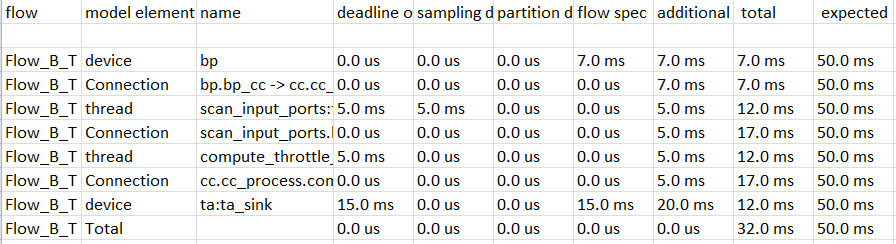
We have previously analyzed the end to end flows for the car system, when only the cruise control and the context components were defined, and when we elaborated cruise control into functions. Now we run the latency analysis to see how the task architecture, i.e., the threads, affect the end to end latency. In the case of periodic threads there is sampling latency and there is computational latency. In the analysis the worst case computational latency is assumed to be the thread deadline, i.e., the threads are schedulable. Note that the latency is also affected by whether the connections are sampling connections, immediate (mid-frame) connections, or delayed (frame-delayed) connections.

To run the analysis we first instantiate the top level system implementation, i.e., the implementation of MyCar. Then we select the instance model file and invoke the *Flow Latency* analysis as instructed in the previous session.

The result of the analysis is shown below. Note that in the Asynchronous case the latency is the same as for the synchronous case.



When we examine the report spreadsheet we get a little more insight into how the latency has been determined.



The spread sheet shows the synchronous scenario of *Flow\_B\_T*. We see the 7 ms of bp, then a sampling latency of 5ms. This is due to the fact that the device operates aperiodically and the first thread samples periodically at a period of 5 ms. The three threads are chained together with immediate connections, i.e., they execute in order within the same frame and they have to complete by the deadline of the last thread, which is 5 ms. Finally we have the latency added by the throttle actuator.

In the asynchronous case, without explicit binding to processors, we assume that these threads reside in processor(s) within the same time domain in order to ensure the specified *immediate* connection timing. This is because the threads are connected by immediate connections.

To gain a better insight as to how the flow latency plugin works, please refer to the SEI Technical report: Feiler, Hansson, “Flow Latency Analysis with the Architecture Analysis and Design Language,” CMU/SEI-2007-TN010, 2007. located on the course DVD or here: <http://www.sei.cmu.edu/library/abstracts/reports/07tn010.cfm>

### Weight Analysis Revisited

Note that at any time we can revisit other analyses. Since we do have weight information in the model we can ensure that our design changes have not affected the total weight of the system. Since we have only elaborated software components we should not get any changes to the total weight.