Session 5: Automotive Systems Modeling

Flow Modeling

This is a hands-on exercise to become familiar with specifying flows in high-level models.

### Goals of the Exercise:

* Reinforce the understanding of flow specifications, implementations, and end-to-end flow declarations.
* Model two flow paths of interest by augmenting cruise control and context components with the appropriate flow specifications.
* Run the latency analysis on the flows of interest and modify the latency values to correct a condition where the actual latency was greater than the latency specification for a particular flow.
* Identify and model software subcomponents of the cruise control system.
* Add flow information to those subcomponents.
* Rerun the latency analysis on the expanded model.

### Setting up the exercise:

We will begin with the completed model from Session 3. To ensure that everyone starts with the same model, we will import a completed model from the DVD. The steps are:

1. Start OSATE and switch to a new workspace my\_session5\_workspace. This can be done by File -> Switch Workspace -> Other and typing in the path and the workspace name, e.g. <prefix>/my\_session5\_workspace.
2. Import the model created in Session 3 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 3/session3\_solution.zip. <prefix> is the path to the session3\_solution.zip file to be imported. Click finish and the Session 3 project will appear in the AADL Navigator.

### Problem Scope:

The requirements of embedded real-time systems specify that certain actions within a system must complete within a specific absolute value or time, or within a time range, with the upper limit as an absolute value. For example, in an avionics system, when a hardware button on the cockpit display is pressed to initiate a screen change, the new screen (with all the relevant data) must be displayed within 50 ms. Other types of actions, for example, are safety critical. When the yoke of an aircraft is turned or pushed in, the control surfaces should receive their control signal updates between 50 ms and 60 ms. Such timing is critical for control laws to maintain the stability of the aircraft.

In this exercise, we will look at similar timing requirements for the cruise control system at a fairly high level of abstraction, specifically by looking at the cruise control software as a whole and at a course-grain decomposition of the software. In a later exercise, we will model the software subcomponent runtime characteristics (e.g. threads) and consider the architectural decisions in going from a data/control flow view to a runtime view.

### Problem Details:

**Initial End to end Flow Latency Analysis**

As an embedded systems engineer in an automotive company R&D organization, you have been given the task to modify existing cruise control software to add filtering to some of the sensor signal because the control engineer has determined that noise causes the cruise control to sometimes increase speed when transitioning from disengage to engaged. In addition, you are to ensure that the end-to-end computational latency of the system is within the specified requirements. In order to do this, you decide to use AADL to model the current software and sensor and actuator communication characteristics to identify the data and event flows, and to understand their data dependencies and execution flow. Having previously developed a component-connector model of the MyCar system, you can now add latency properties to the components and perform analysis on the model to validate the end-to-end latency.

You also make some observations that the inputs and outputs are from devices that you have modeled in the previous exercise, and that their latencies are documented in their data sheets.

You have been told that flow from the brake pedal to the throttle actuator is safety critical and you must ensure that the end-to-end latency from the brake pedal to the throttle actuator is less than or equal to the latency specified in the system requirements document. In addition, since you are adding filtering in the control law section, that flow from the wheel to the throttle actuator is important for the stability of the controller. The latency for this flow must comply with that specified in the system requirements document.

Therefore, the two flows of interest are:

* Flow Wheel to Throttle (Flow\_W\_T): Wheel rotation pulses through the cruise controller control laws to the signal at the throttle actuator.
* Flow Brake Pedal to Throttle (Flow\_B\_T): Brake pedal depression through the cruise control mode logic to the throttle actuator.

### Latencies

The required end-to-end latency from the brake pedal to the throttle actuator is 50 ms. The required end-to-end latency from the wheel rotation sensor to the throttle actuator is 40 ms.

Latencies for the device and system components are given below. During this phase of modeling, these latency values are *estimates*. To be realistic, measurements will need to be made on the actual running code. (NOTE: these latency values are defined for the purposes of this exercise and do not reflect any specific automotive system.)

* Flow\_W\_T for Cruise Control: wrs input to ta output is 20 ms.
* Flow\_B\_T for Cruise Control: bp input to ta output is 15 ms
* Flow from the wheel rotation sensor: F\_WRS, Latency value of 5 ms.
* Flow from the brake pedal F\_BP. Latency value of 7 ms.
* Flow to the throttle actuator: F\_T. Latency value of 15 ms.

**IMPORTANT NOTE**: Flow latency values are specified as a range, e.g. 10ms..10ms.

**Detailing the Functions of the Cruise Control System**

Next, you can add the detailed software components within the cruise control application and associated latency properties to the components and perform analysis on the model to revalidate the end-to-end latency.

To understand the existing design and implementation, you review software design documents but find them somewhat incomplete. Through conversations with colleagues in the group, you find out that the software has been modified several times and that the design documents may not accurately represent the architecture that is currently implemented in the firmware. You decide to go to the source code, read through it, and develop a picture of the software components and how they interrelate.

You determine that there are four software subcomponents (modules) that comprise the cruise control software:

* Scan\_cc\_inputs: performs a periodic reading of inputs from various sensors and other software systems and determines the state of the vehicle with respect to the functions of the cruise control.
* Speed\_control\_loop: contains a speed control law, a version of a Proportional Sum Difference (PSD) controller. It uses the error between the wheel rotational speed and the desired speed setpoint to determine the position of the throttle actuator. You also notice something interesting that the speed setpoint is obtained by saving the current wheel velocity at the time an ok\_to\_run variable is set by the scan\_cc\_inputs.
* Compute\_throttle\_setting: performs the scaling of speed setpoint, provides some range checking, and then outputs the binary value to and D/A that is part of the throttle actuator.
* Compute\_velocity: counts the wheel pulses per unit of time and determines the current instantaneous velocity.

### Modeling Approach

Developing a model that depicts software data and communication flow is best done initially using AADL systems, ports and connections (e.g. a component-connection view). The reasons for approaching the modeling in this manner include: each software component can be represented at various hierarchical levels (and detailed later as needs arise); data can be abstracted to meaningful names (and can be detailed in later refinements, as necessary); and a architect can focus on the logical flow of data and events, while leaving additional architectural decisions such as mapping to threads and execution platform deployment until later.

To help with consistency of models developed by all class participants, the following feature group or port names should be used:

* For ScanCCInputs, the inputs are: brake\_status, wheel\_pulse, set\_speed, accelerate, coast, engine\_status, cc\_system\_on\_off. The outputs are: run\_cc.
* For SpeedControlLoop, the inputs are: run\_cc, inst\_velocity. The outputs are: desired\_speed
* For ComputeThrottleSetting, the inputs are: desired\_speed. The outputs are: throttle\_setting.
* For ComputeVelocity, the inputs are: wheel\_pulse. The outputs are: inst\_velocity.

For modeling completeness the above software components will be contained in a software system called *CruiseControl*.

### Latencies within Cruise Control

Latencies for the cruise control functions are estimations or assigned latency budgets. (NOTE: these latency values are defined for the purposes of this exercise and do not reflect any specific automotive system.)

* Flow for ScanCCInputs: brake\_status to run\_cc is 15 ms.
* Flow for SpeedControlLoop: run\_cc to desired\_speed is 10 ms, inst\_velocity to desired\_speed is 5 ms
* Flow for ComputeThrottleSetting: desired\_speed to throttle\_setting is 5 ms
* Flow for ComputeVelocity: wheel\_pulses to inst\_velocity is 5 ms.

**IMPORTANT NOTE**: Flow latency values are specified as a range, e.g. 10ms..10ms.

Construct a software component and connection model, using the component names above. Include the relevant sensors, actuators, switches, etc. in the model.

### Construction of the model

You now have enough information to begin modeling. It is suggested that you initially make a sketch of the components and the data/event flow connections. From there, a model can be created textually or through the graphical editor. Note that flow information cannot be entered through the graphical editor and will have to be added using the text editor.

### Setting up the initial model

If you have not already done so, create a workspace and import the Session 3 model as described on page 1, “Setting up the exercise.”

### Adding Flow Specifications and Latencies to Components

In developing the flow analysis model, we will work with the AADL textual and graphical files and will use the components and packages defined for the model in Session 3. First, you should add the flow specifications and latency properties to the Type declarations for the Devices:

* *BrakePedal* and *WheelRotationSensor* in the *Sensors.aadl* package
* *ThrottleAcutator* in the *Actuators.aadl* package

and to the System Type:

* *CruiseControl* in the package *SoftwareApps*.
* There should be two flow path specifications for the *CruiseControl* system, one for each of the flows: Wheel Sensor to Throttle (*Flow\_W\_T*) and Brake Pedal to Throttle (*Flow\_B\_T*).

To add flow specification entries, you use the AADL text editing capabilities of OSATE. To do this:

* Open the appropriate package file in the aadl folder (e.g. *Sensors*) by double clicking or right click and select Open in the resulting pop-up menu or left double click on the aadl package name.
* Enter the flow specifications and latencies in the Type declarations for each of the components.

An example flow specification for a *WheelRotationSensor* Type declaration is shown in the listing below. This is a flow source specification, indicating that an end-to-end flow can originate from the component. Note that your naming convention for some of the elements may differ from the examples shown.

|  |
| --- |
| **device** WheelRotationSensor  **features**  wrs\_cc: **port** **group**;  wrs\_abs: **port** **group**;  **flows**  wrs\_source: **flow** **source** wrs\_cc {Latency => 5 ms;};  **end** WheelRotationSensor; |

The BrakePedal device Type in the Sensors package will have a similar flow source statement. The ThrottleActuator device Type in the Actuators package will be similar except that is it a flow sink with 15 ms latency.

An example for the *CruiseControl* Type in the SoftwareApps package, which has two flows, is shown below.

|  |
| --- |
| **flows**  -- defines a wheel sensor to throttle flow path  Flow\_W\_T: **flow** **path** cc\_wrs -> cc\_ta{  Latency => 20 Ms .. 20 Ms;  };  -- defines a brake pedal to throttle path  Flow\_B\_T: **flow** **path** cc\_bp -> cc\_ta{  Latency => 15 Ms .. 15 Ms;  }; |

This completes adding the flow source, sink, and path to the appropriate component types.

Next, you need to define the end-to-end flows: Wheel Sensor to Throttle (*Flow\_W\_T*) and Brake Pedal to Throttle (*Flow\_B\_T*), in *MyCar.impl* that is contained in the *CarSystem* package. An example, of these declarations is shown below.

|  |
| --- |
| **flows**  Flow\_B\_T: **end** **to** **end** **flow** bp.bp\_source -> bp\_cc\_conn -> cc.Flow\_B\_T  -> cc\_ta\_conn -> ta.ta\_sink  {  Latency => 50 Ms .. 50 Ms;  };  Flow\_W\_T: **end** **to** **end** **flow** wrs.wrs\_source -> wrs\_cc\_conn -> cc.Flow\_W\_T  -> cc\_ta\_conn -> ta.ta\_sink  {  Latency => 40 Ms .. 40 Ms;  }; |

### Running the Latency Analysis

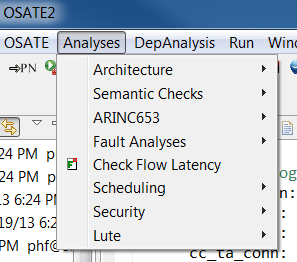
### In the outline view select your Mycar implementation and create an instance model by invoking the appropriate command in the context menu.

Next select the resulting instance model in the *instances* folder. This enables analysis commands on the instance model.

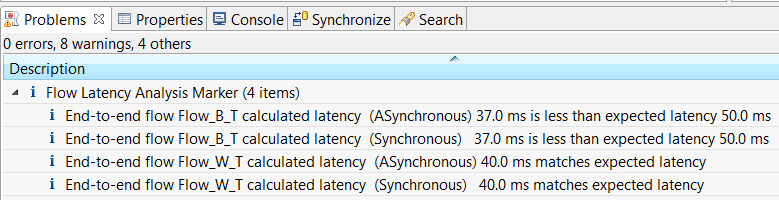
To run the flow analysis, you select the instance file in the AADL Navigator and invoke the analysis on that file in one of three ways:

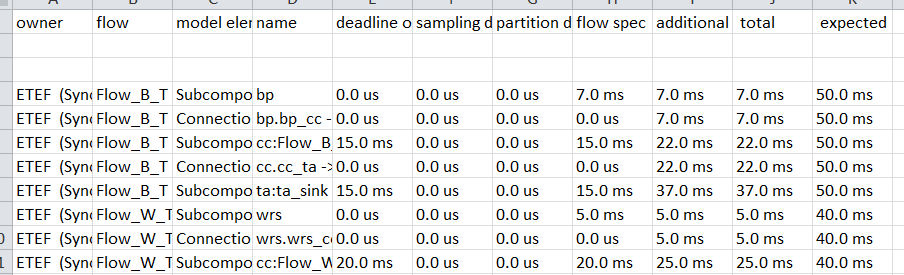
* Context menu: right click on the file and select AADL Analysis -> Check Flow Latency
* Analyses menu: Pull down the Analyses menu in the Menu bar and select “ Check Flow Latency.”
* Toolbar icon: Click the flow analysis icon ~MyBitmap.

*Check Flow Latency* in the *Analyses* menu:



The analysis will be run and the results presented in the Problems view. You will see the results for both end to end flows. In addition you will see each flow analyzed twice: once on the assumption that the software executes on a hardware platform that operates on a single clock (synchronous); and once where each software component is assumed to operate on a processors, each on its own clock (asynchronous). This second figure may be higher to reflect the fact that clocks may be misaligned and clock drift may occur - resulting in increased latency.

They will also be reported in an Excel file in the *reports* folder. This spreadsheet details out the way the latency has been calculated. It will help you understand the result, if you expected different numbers.



### Detailing the Cruise Control System

At this point we must develop the cruise control software system model by defining the constituent software components (using the textual or graphical interfaces), include instances of those subcomponents in the cruise control implementation, then add the flow specifications and latencies for each subcomponent.

Begin by creating a SoftwareParts package which will hold the software components to be contained in the cruise control:

* Create the *SoftwareParts.aadl* file in the AADL Navigator, and add the package declaration into the file.
* Use the create **system** type declarations for each of the four components specified below.
  + ScanCCInputs, the inputs are: **brake\_status**. The outputs are: **run\_cc.**
  + For SpeedControlLoop, the inputs are: **run\_cc**, **inst\_velocity**. The outputs are: **desired\_speed**
  + For ComputeThrottleSetting, the inputs are: **desired\_speed**. The outputs are: **throttle\_setting**.
  + For ComputeVelocity, the inputs are: **wheel\_pulse**. The outputs are: **inst\_velocity**.
* Now add the flow information with latency data as shown below. Note that two are required (Flow\_B\_T and Flow\_W\_T) for the *SpeedControlLoop* component, since each involves a different latency.

**system** SpeedControlLoop

features

run\_cc: **feature** **group**;

inst\_velocity: **feature** **group**;

desired\_speed: **feature** **group**;

flows

Flow\_B\_T: **flow** **path** run\_cc -> desired\_speed {

Latency => 10 Ms .. 10 Ms;

};

Flow\_W\_T: **flow** **path** inst\_velocity -> desired\_speed {

Latency => 5 Ms .. 5 Ms;

};

**end** SpeedControlLoop;

* Save the file with these additions.

Now we use those system types to define the implementation of cruise control.

* Create a CruiseControl implementation by opening SoftwareApps .aadl and then adding a system implementation (remember <ctrl><space> after typing “sy” to get the AADL template).
* Add *SoftwareParts* to the *with* clause of *SoftwareApps*.

Add in the four subcomponents referring to the system types (ScanCCInputs, SpeedControlLoop, ComputeThrottleSetting, ComputeVelocity).   
 **system** **implementation** CruiseControl.impl

subcomponents

sci: **system** SoftwareParts::ScanCCInputs;

cv: **system** SoftwareParts::ComputeVelocity;

scl: **system** SoftwareParts::SpeedControlLoop;

cts: **system** SoftwareParts::ComputeThrottleSetting;

* Add the connections as shown below.

**connections**

bp\_sci\_conn: **feature** **group** cc\_bp <-> sci.brake\_status;

wr\_sci\_conn: **feature** **group** cc\_wrs <-> sci.wheel\_pulse;

wr\_cv\_conn: **feature** **group** cc\_wrs <-> cv.wheel\_pulse;

cv\_scl\_conn: **feature** **group** cv.inst\_velocity <-> scl.inst\_velocity;

sci\_scl\_conn: **feature** **group** sci.run\_cc <-> scl.run\_cc;

scl\_cts\_conn: **feature** **group** scl.desired\_speed <-> cts.desired\_speed;

cts\_ta\_conn: **feature** **group** cts.throttle\_setting <-> cc\_ta;

* Add the flow path implementation information into the cruise control implementation. It specifies how the flow specifications in the cruise control type will be expanded when an instance model is created.  
   **flows**

Flow\_B\_T: **flow** **path** cc\_bp -> bp\_sci\_conn -> sci.Flow\_B\_T

-> sci\_scl\_conn -> scl.Flow\_B\_T

-> scl\_cts\_conn -> cts.Flow\_1

-> cts\_ta\_conn -> cc\_ta;

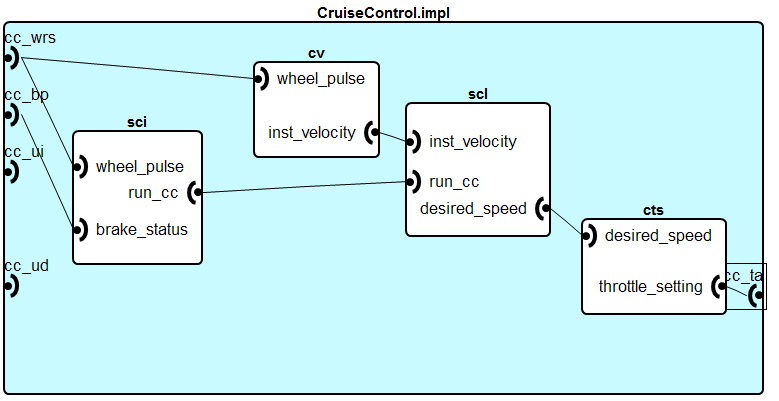
Flow\_W\_T: **flow** **path** cc\_wrs -> wr\_cv\_conn -> cv.Flow\_W\_T

-> cv\_scl\_conn -> scl.Flow\_W\_T

-> scl\_cts\_conn -> cts.Flow\_1

-> cts\_ta\_conn -> cc\_ta;

* Save the file. Select the *SoftwareApps.aadl* file in the Navigator.
* Invoke the Graphical Model Viewer from the *Osate* entry in the context menu. You should get a graphical view that you can arrange as shown below.



### Rerunning the Latency Analysis

First, we need to edit the CarSystem package to configure MyCar to make use to the cruise control implementation we just created.

|  |
| --- |
| cc: **system** SoftwareApps::CruiseControl.impl; |

Note: we can create a second cruise control implementation that contains the above expanded declaration for cc. This allows us to maintain both the original and the expanded model as separate instance models to compare the results. In a later session we will see how to make use of the **extends** and **refined to** constructs to explicitly represent the refinement step in the AADL model.

Now we select the implementation in the *Outline* view and create the instance model again.

Then we rerun the latency analysis on the refined instance model and get the result below. Note that for the end-to-end flow *Flow\_B\_T* exceeds the expected value. Examining the report spreadsheet we can see that the latencies for the functions within cruise control result in a higher latency than the originally specified latency on the cruise control flow paths.

|  |
| --- |
|  |

Note that the latency analysis reflects how a control engineer typically views latency. It is based on the amount of type it takes to execute a function and the time it takes to communicate the signal through the sensors and actuators. At this time we consider any latency due to communication between the device and the cruise control subsystem negligible. However, as necessary we can add latency numbers to the connections to take such communication latency into account.

When we compare the latency results to those run before we expanded the model, we see that for *Flow\_W\_T* the latency became smaller. This indicates that we are within the latency budget assigned to this flow through the cruise control subsystem. At the same time, *Flow\_B\_T* has a larger number and exceeds the end to end latency requirement.

### Exploring Alternatives

To remedy a situation where the actual latency for the brake to throttle actuator flow is greater than the expected latency, you can explore alternatives for the various latency values associated with the devices and software within the system. For example, after discussing the latency issue with control and hardware engineers, a possibility to reduce the brake pedal latency to 5 ms using an alternative design is suggested. They are checking the cost and feasibility of doing so. In the meantime, re-run the analysis to verify that such a change reduces the actual latency to less than or equal to the expected latency.