Collins CASE TA2 Monitor Transform

|  |
| --- |
| To illustrate the Monitor transform, we use a simple producer-consumer example model, which can be found here:  https://github.com/loonwerks/CASE/tree/master/TA2/Model\_Transformations/Monitor/Simple\_Example  Two AADL packages are included:   * Monitor.aadl – This is the initial model. * CASE\_Requirements.aadl – This is the package containing the requirement (in the form of a Resolute claim) that drives the monitor transform.   The Monitor transform can also be performed on the CASE Phase 1 UAV example model. Three versions of the model are available for reference:   * Initial model – This is the Phase 1 UAV model that includes an imported cyber requirement, which drives the attestation of the source of incoming messages to the FlightPlanner component. The Initial model can be found here:   [https://github.com/loonwerks/CASE/tree/master/TA2/Model\_Transformations/Monitor/ UAV\_Example/Initial\_Model](https://github.com/loonwerks/CASE/tree/master/TA2/Model_Transformations/Monitor/%20UAV_Example/Initial_Model)   * Transformed model – This is the Phase 1 UAV model after the Monitor transform has been applied. The Transformed model can be found here:   [https://github.com/loonwerks/CASE/tree/master/TA2/Model\_Transformations/Monitor/ UAV\_Example/Transformed\_Model](https://github.com/loonwerks/CASE/tree/master/TA2/Model_Transformations/Monitor/%20UAV_Example/Transformed_Model)   * Test model – This is the Phase 1 UAV model containing several software implementations for testing the correctness of the Resolute evaluation on the Monitor transform. The Test model can be found here:   [https://github.com/loonwerks/CASE/tree/master/TA2/Model\_Transformations/Monitor/ UAV\_Example/Test\_Model](https://github.com/loonwerks/CASE/tree/master/TA2/Model_Transformations/Monitor/%20UAV_Example/Test_Model) |

A CASE Monitor is added to an AADL connection to provide an alert when a policy on the connection’s data has been violated (or alternatively, satisfied). In addition to signaling an alert when the monitor policy has been violated, the monitor can also be used as a gate to prevent data from propagating. Multiple monitors may be placed on the same connection. The Monitor component type that is inserted into the model will be the same component type as the connection source, with two exceptions: (1) If the destination component is a thread group, the filter will be a thread. (2) If the destination component is a process containing a single thread, the filter will also be a process containing a single thread. The latter supports the seL4 representation of components, in which each thread runs in its own address space. Note that for System Build, the filter must be a software component (either a thread or process containing a single thread).

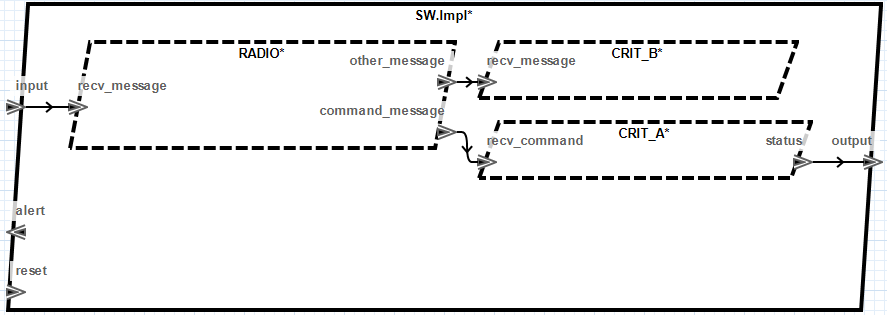


Figure 1. Initial model.

A monitor can be added to the following AADL components:

* Thread
* Thread Group
* Process containing a single thread – Not yet implemented

To insert a monitor, select the connection in a component implementation that requires monitoring (for example, in Monitor.aadl, select the c1 connection on line 55). Note that currently the transform can only be applied from within the OSATE text editor (future versions will enable applying the transformation from within the graphical editor). In the main menubar, click the CASE 🡪 Cyber Resiliency 🡪 Model Transformations 🡪 Add Monitor… menu item. A wizard will open, as shown in Figure 2. The wizard enables the user to customize the monitor, including providing the monitor policy specification.

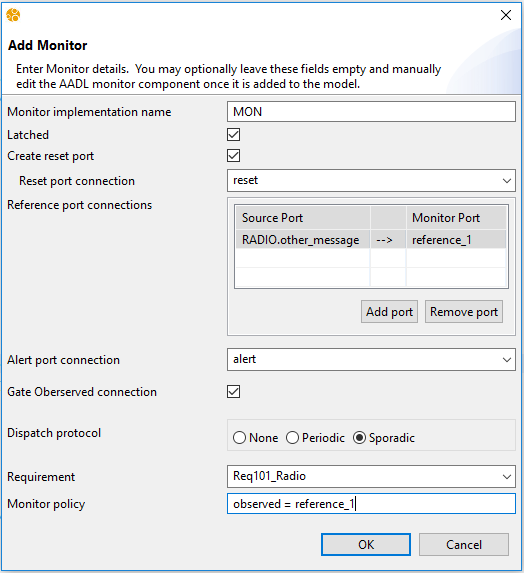


Figure 2. Add Monitor wizard.

The Monitor transform will create a special CASE\_Monitor AADL component type and implementation, and insert them into the model. It will then instantiate the CASE\_Monitor implementation as a subcomponent in the implementation containing the selected connection. The user may provide a name for the monitor subcomponent, or use the default. If the field is left blank, the default name will be used. Note that if the specified name already exists, a number will be appended to the name to make it unique within the containing component implementation.

Runtime monitors can be *latched*, meaning once the monitor policy has been violated the alert signal will remain high until the monitor is *reset*. A reset port can be added to the component by checking the box. This will enable the dropdown box containing other ports in the model with which the monitor reset port can be connected. If a source reset port is selected, the monitor reset port will be of the same type (Data, Event, Event Data). It is not necessary to select a specific reset source port to add a reset port.

Monitors typically function by comparing an observed signal with one or more reference signals. The *Reference port connections* input provides the ability to create monitor reference ports, select a source component port to connect to, and provide a name for the reference port.

The *Alert port connection* dropdown box contains other ports in the model to connect the monitor alert port to. The monitor alert port will be the same type as the destination alert port. It is not necessary to select a specific alert destination port, in which case the monitor alert port will be an event data port with no data type.

By selecting the *Gate Observed connection* checkbox, a monitor will be generated that does not only listens on a specific connection, but routes the connection through the monitor. In this configuration, when the monitor policy is violated, the monitor can prevent the propagation of the observed signal.

The requirement drop-down box lists all of the cyber-requirements that have been imported from TA1 tools. By specifying the cyber requirement that drives the attestation transform, the appropriate assurance argument can be constructed for demonstrating the requirement was addressed correctly. A requirement does not need to be selected to insert the monitor, but it is highly recommended for construction of the proper system assurance case.

Finally, the user may provide the *Monitor policy*, in the form of an AGREE statement. The policy should evaluate to a boolean value, and typically compares the observed signal to constant values or reference signals. Note that no syntax validation is performed on the AGREE statement. If it is malformed, it may not be imported into the model properly.

Clicking the OK button on the wizard will insert the CASE\_Monitor into the model, as shown in Figure 3 and Figure 4.

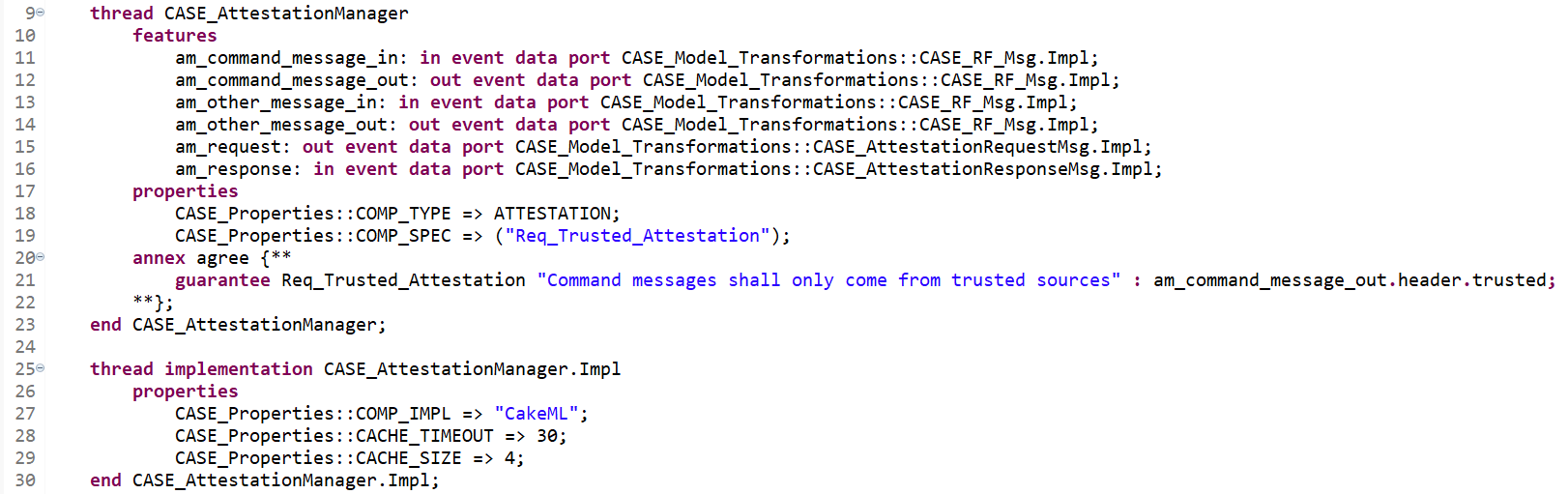


Figure 3. Line 9: CASE\_Monitor component type; Line 25: CASE\_Monitor component implementation.

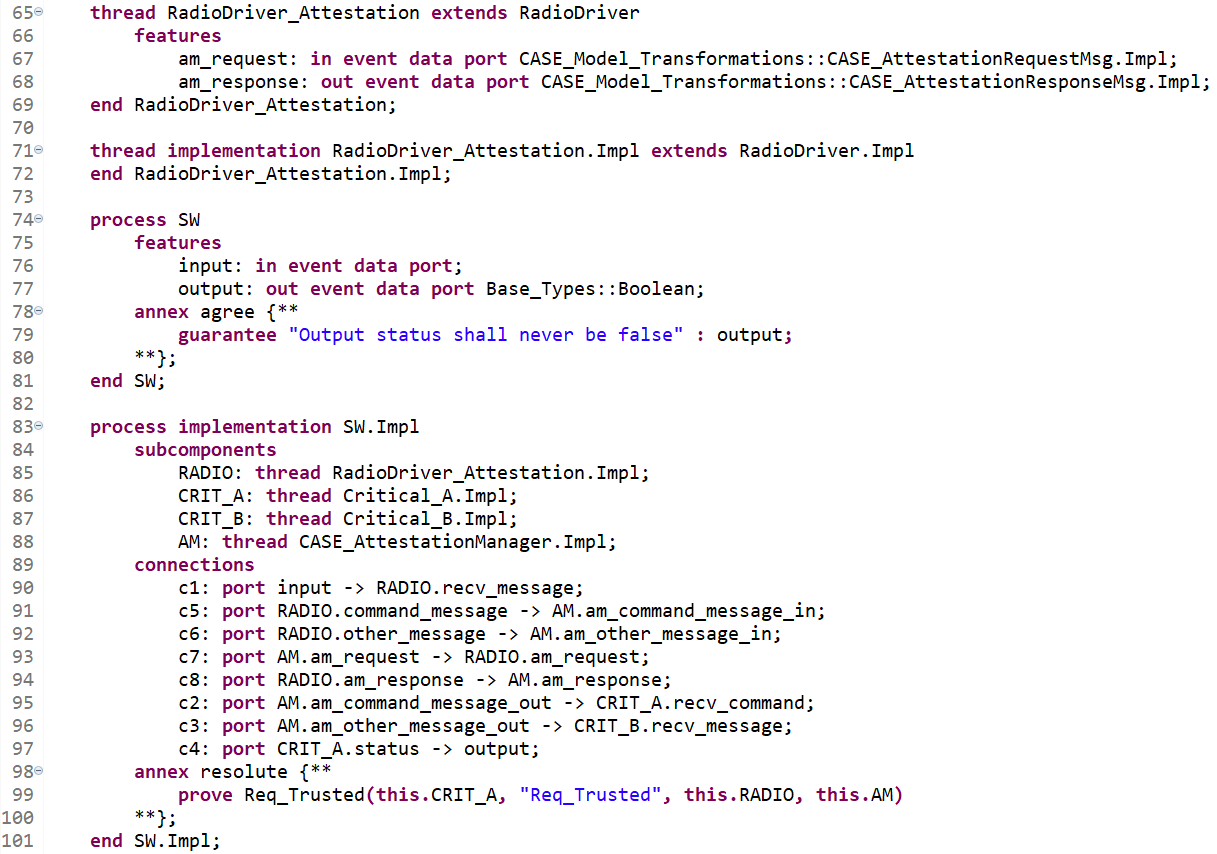


Figure 4. Line 88: Monitor subcomponent; Lines 91-96: Monitor connections; Line 99: updated assurance claim call.

### Design Assurance

It is crucial to have evidence of design correctness both at the time the model transformation is performed, and at any time up through system build. Resolute provides that assurance via augmentation of the requirement with assurance subclaims as model transformations are performed.

When a requirement is imported from a TA1 tool it will be placed in the CASE\_Requirements package as a Resolute claim, as shown in Figure 6.

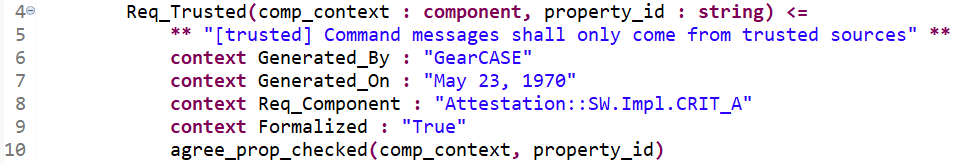


Figure 6. Requirement imported from a TA1 tool.

Initially, there is not much for Resolute to check because the requirement hasn’t yet been addressed in the design. All Resolute can do in this example is check that AGREE analysis was performed. Note that Resolute uses a separate plugin called AgreeCheck to determine if AGREE analysis was performed. AgreeCheck is included with Resolute, but requires initial user configuration. In order to successfully use AgreeCheck, "Generate property analysis log" must be checked in the AGREE Analysis preferences, and a log file pathname must be specified. The AGREE Analysis preferences can be accessed by selecting Window → Preferences from the main menu, expanding the Agree node on the left-hand side of the preference window, and selecting Analysis.

Once the Attestation transform is applied, the requirement is updated with an additional check to make, which reflects the addition of the attestation manager component, as shown in Figure 7.

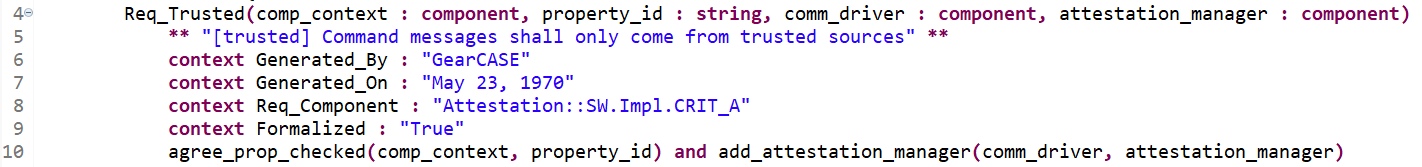


Figure 7. Modified requirement after Attestation transform.

The addition of the add\_attestation\_manager() call on line 10 provides Resolute with additional checks to make to ensure the requirement was addressed correctly. In this case, add\_attestation\_manager() is included in the CASE\_Model\_Transformations library and consists of two subclaims:

* attestation\_manager\_exists() – Checks that the attestation manager component is present in the model
* attestation\_manager\_not\_bypassed() – Checks that there are no connections in the model that bypass the attestation manager

To check whether the requirement has been correctly addressed in the design, select the containing component implementation (SW.Impl on line 83 in Attestation.aadl) and select Analyses 🡪 Resolute from the main menubar. The Resolute output will appear in the output pane, as shown in Figure 8.

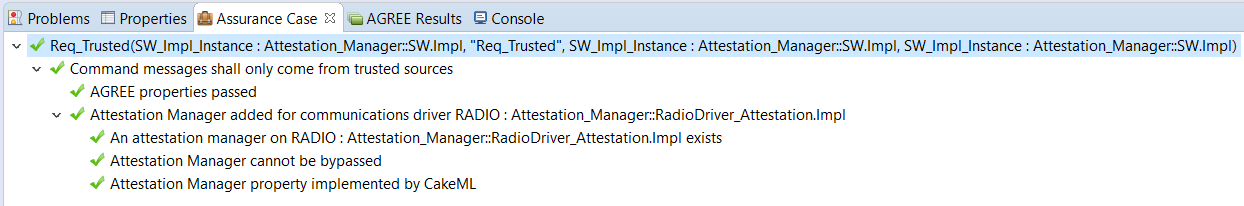


Figure 8. A passing Resolute analysis.