Collins CASE TA2 Filter Transform

|  |
| --- |
| The Filter transform can be performed on a simple UAV example model. Two versions of the model are available for reference:   * Initial model – This is the simple UAV example model that includes an imported cyber requirement, which drives the well-formedness of incoming messages to the FlightPlanner component. The Initial model can be found here:   <https://github.com/loonwerks/CASE/tree/master/TA2/Model%20Transformations/Filter/Initial%20Model>   * Transformed model – This is the simple UAV example model after the Filter transform has been applied. The Transformed model can be found here:   <https://github.com/loonwerks/CASE/tree/master/TA2/Model%20Transformations/Filter/Transformed%20Model> |

A CASE Filter is added to a component’s input port to ensure that only data that matches a specified regular expression arrives on that input port. To add a filter to the model, a connection must be selected that terminates at the input port of a component. For example, Figure 1 shows a thread subcomponent connected to its parent by connection c1. Also in the figure, connection c2 connects two thread components. A filter can be inserted onto either of these connections. However, a filter cannot be inserted onto connection c3, which connects the component to its parent. This is because a filter is always associated with the *input* port of a component.

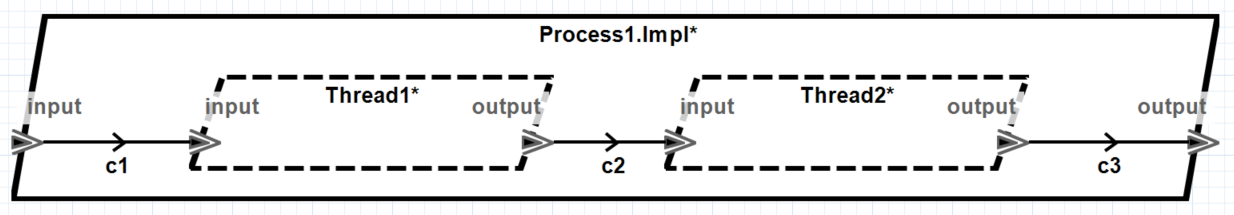


Figure .

A filter can be added to the following AADL components:

* Abstract
* System
* Device
* Process
* Thread Group
* Thread

The model transform will be insert a filter component that has the same type as the component it connects to, 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).

To insert a filter, select the connection in a component implementation that terminates at the component that requires filtered input (for example, in SW.aadl, select the c3 connection on line 224). Note that currently the transformation 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 Filter… menu item. A wizard will open, as shown in Figure 2. The wizard enables the user to customize the filter, including providing the filter specification.

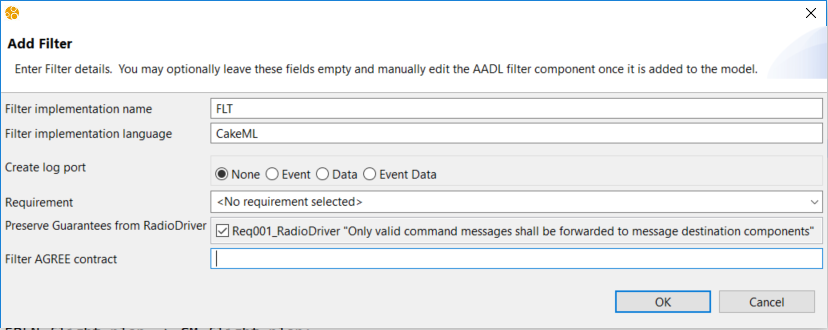


Figure . Add Filter wizard

The Filter transform will create a special CASE\_Filter AADL component type and implementation, and insert them into the model. It will then instantiate the CASE\_Filter as a subcomponent in the selected implementation. The user may provide a name for the filter 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.

By default the CASE filter will drop any messages that do not match the regular expression and no record of the malformed message will be retained. If the user wishes to log the event, an additional log port can be added to the filter. The user will need to specify the AADL port type (Event, Data, or EventData) and it will be up to the user to connect the log port to an appropriate “logger” component.

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 filter transformation, 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 filter, but it is highly recommended for construction of the proper system assurance case.

Finally, the user may provide the filter specification as an AGREE *guarantee* statement. This is typically done by referring to the outgoing message on the filter’s output port. The message type will be the same as the target component’s input port. Within the AGREE statement, the filter output port name by *filter\_out*. For example, if the message type is a signed integer, and the filter should drop any message with a value less than zero, the AGREE statement will be:

guarantee Req001\_Filter "Only non-negative numbers" : filter\_out >= 0;

In the UAV example, the filter expression is expressed in AGREE by the function good\_map() (SW.aadl, line 252), and so the filter AGREE statement will be:

guarantee Req001\_Filter "The FlightPlanner shall only receive well-formed maps" : good\_map(filter\_out.payload.map);

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\_Filter into the model, as shown in Figure 3 and Figure 4.

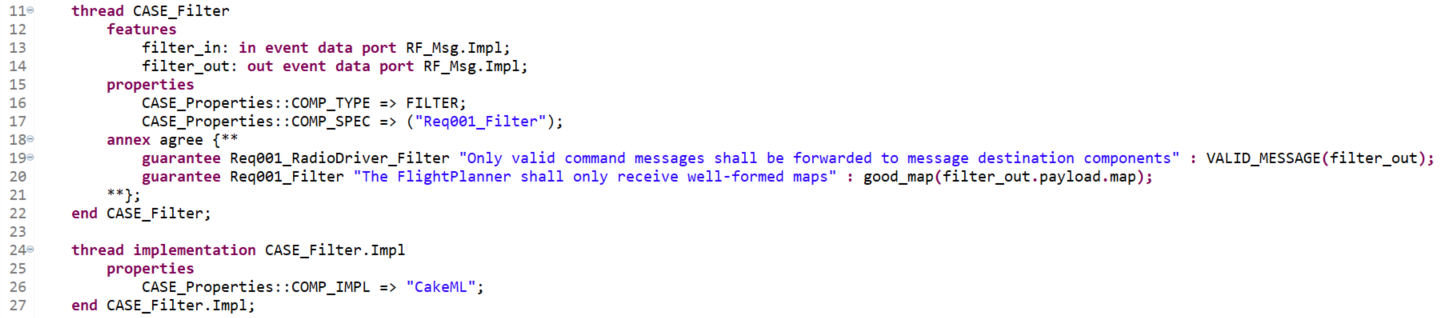


Figure . Line 11: CASE\_Filter component type; Line 24: CASE\_Filter component implementation.



Figure . Line 240: filter subcomponent; Lines 244-245: filter connections; Line 262: updated assurance claim.

### Compound Filters

Two CASE filters cannot be connected to each other. Attempting to place a filter on a connection that already has a filter component as either its source or destination will result in a warning, and an option to create a compound filter, as shown in Figure 4. A compound filter is simply a single filter component containing multiple filter expressions.

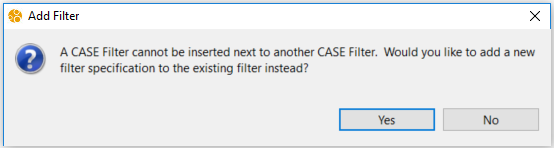


Figure . Adjacent CASE Filter warning.

Selecting “Yes” will display a wizard similar to adding a new filter, however, some fields are disabled since only a new filter spec is being added (see Figure 6). The result of creating a compound filter is an additional AGREE guarantee statement in the filter component. The conjunction of these specifications describe the filter.

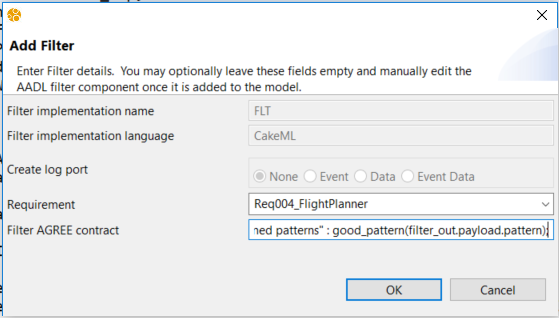


Figure . Adding an expression to an existing filter.

For example, adding the additional filter expression:

guarantee Req002\_Filter "The FlightPlanner shall only receive well-formed patterns" : good\_pattern(filter\_out.payload.pattern);

The new expression will appear in the existing filter as shown in Figure 7.

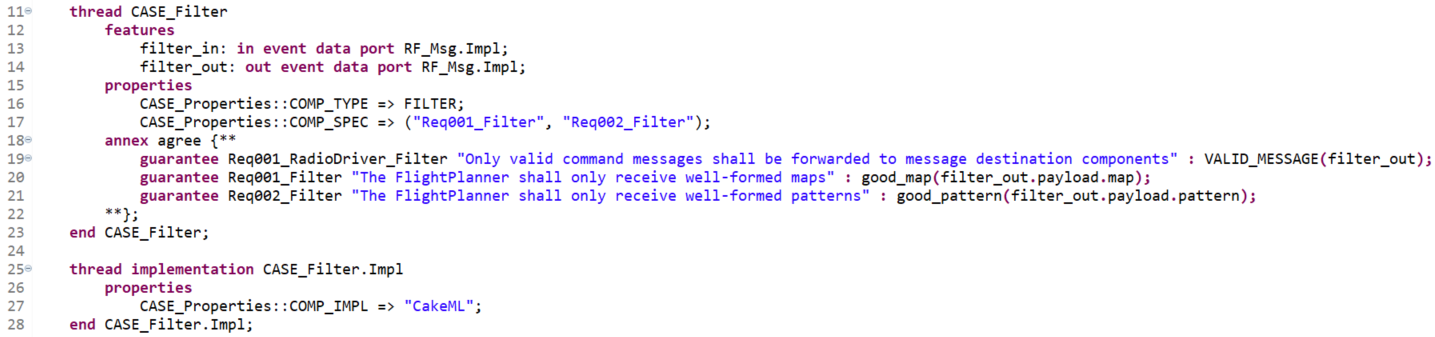


Figure . Compound filter.

### CASE Filter Properties

If an implementation language is specified, it will be included as the CASE\_Properties::COMP\_IMPL property association on the filter implementation (line 27 in Figure 7). For filter synthesis using SPLAT, two other properties are necessary. The CASE\_Properties::COMP\_TYPE => FILTER property association indicates that the component is a CASE Filter. The CASE\_Properties::COMP\_SPEC property association lists the AGREE specification IDs of the guarantee statements that comprise the filter expression. For example, in Figure 7, the COMP\_SPEC property lists two identifiers corresponding to the two AGREE guarantee statements on lines 20 and 21. The specification of these two statements, good\_map() and good\_pattern(), provide the definition of well-formedness for the filter.