Resolint User's Guide

Introduction

Resolint is an OSATE-based linter tool for AADL models. Resolint provides a language for specifying rules that correspond to modeling guidelines, as well as a checker for evaluating whether a model complies with the rules. Results of the Resolint analysis are displayed to the user. Rule violations will indicate severity, and are linked to the model element that is out of compliance with the rule.

How to Get Resolint

Resolint is currently included with the Resolute plugin for OSATE, which can be found here:

https://github.com/loonwerks/formal-methods-workbench/tree/master/tools/resolute

Relationship with Resolute

Resolint statements are contained in the Resolute annexes of AADL models. This is because Resolint rules are specified using the same grammar as Resolute claims. In addition, the Resolute evaluation engine is used to determine whether the AADL model is in compliance with the Resolint rules. Otherwise, Resolint and Resolute are two different tools with two different use cases. Future versions of Resolint may have greater independence from Resolute.

Formalizing Rules in Resolint

Rules derived from sources such as development standards, checklists, and modeling guidelines can be encoded in Resolint and embedded in a Resolute Annex in an AADL model. Rules in Resolint are represented using the same syntax as Resolute claims. For more information on specifying claims in Resolute, see the Resolute User's Guide here:

https://github.com/loonwerks/formal-methods-workbench/blob/master/documentation/resolute/user_guide.pdf

For example, the following rule

Threads should have the Dispatch_Protocol property specified

would be specified in Resolint as

```
annex resolute {**

    dispatch_protocol_specified() <=
        ** "Threads should have the Dispatch_Protocol property specified" **
        forall (t : thread) .
            lint_check(t, has_property(t, Dispatch_Protocol))

**};
```

Similarly, the rule

T*hreads can only specify a Dispatch_Property of* Periodic *or* Sporadic

Would be specified as

```
annex resolute {**

    valid_dispatch_protocol() <=
        ** "Threads can only specify a dispatch_protocol property of periodic or sporadic" **
        forall (t : thread) . lint_check(t, has_property(t, Dispatch_Protocol) =>
            (property(t, Dispatch_Protocol) = "Sporadic" or property(t, Dispatch_Protocol) = "Periodic"))

**};
```

Note that each of these rules contains a call to *lint_check()*. lint_check() is a provided function that enables Resolint to capture the specific model element that violates the rule. The definition of lint_check() is specified in Resolint.aadl, which is included with the tool.

```
lint_check(a : aadl, b : bool) <=
    ** a **
    b
```

The function takes an AADL element and a Boolean value. The Boolean value represents the result of the rule check. If it is false, Resolint keeps track of the AADL element that violated the rule in order to provide the user with a direct reference in the results pane.

Two other check functions are provided: lint_check_set() and lint_check_list(). These are used when multiple elements are referenced in a rule. For example, the rule

```
annex resolute {**

    one_process() <=
        ** "Only one processor-bound process can contain thread or thread-group subcomponents" **
        let proc : {process} = {p for (p : process) |
            (exists(pr : processor) . is_bound_to(p, pr)) and forall(s : subcomponents(p)) . (is_thread(s) or is_thread_group(s))};
        lint_check_set(proc, size(proc) = 1)

**};
```

will be violated if multiple process components exist that contain threads or thread groups. If this is the case, the user should be presented with the set of all such processes. lint_check_set() evaluates the size of the set of processes containing threads, and if not equal to 1, will flag all processes in the set.

The lint_check() functions are not necessary for Resolint to check rules and display results. However, they are currently necessary to link results with the AADL elements that are violating the rules. Future versions of Resolint may eliminate the need for the lint_check() functions.

Creating Rulesets

Rules can be grouped into Rulesets. These are useful for organizing rules corresponding to different guidelines, such as organizational process, customer requirements, certification guidelines, and tool constraints. Rulesets also provide the ability to specify the severity of the rule violation; that is, if the rule is found to be violated, what type of message should the user receive. Three levels of severity are supported. From least to most severe, they are *info*, *warning*, and *error*.

The syntax of a ruleset is

```
<Ruleset> ::= 'ruleset' <name> '{' ( <Lint_Statement> )* '}'
```

```
<Lint_Statement> ::= ( ('info' | 'warning' | 'error') <Claim_Function_Reference> )*
```

where <Claim_Function_Reference> is the name of a Resolute claim representing the rule.

Lint Statements are interpreted such that if the Claim Function is false, the user will receive a message marker of the severity indicated by the info, warning, or error keyword.

An example Ruleset is below.

```
annex resolute {**

    ruleset CASE_Tools {

        info (print("Linting CASE_Tools ruleset"))

        -- Threads should have the Dispatch_Protocol property specified
        warning (dispatch_protocol_specified())
        -- Threads can only specify a dispatch_protocol property of periodic or sporadic
        error (valid_dispatch_protocol())

        -- Subcomponent types should be specified
        warning (subcomponent_type_specified())

        -- Array dimensions must be specified
        error (array_dimension())
        -- Arrays can only have one dimension
        error (one_dimensional_arrays())

        -- The array base type should be specified
        warning (array_base_type())
        -- Connections between thread components must be uni-directional
        error (unidirectional_connections())
        -- A processor's subcomponents may be ignored
        warning (no_processor_subcomponents())
        -- AADL modes are not currently supported by CASE tools
        warning (no_modes_in_model())
    }

**};
```

Checking Rules and Rulesets

In order to check that an AADL model complies with a set of rules, Resolint needs to know which rules or rulesets to check. This is specified using the *check()* function call in an AADL component implementation. For example, the *check (CASE_Tools)* call in the UAS.Impl component pictured below will evaluate the UAS.Impl system instance against the CASE_Tools ruleset.

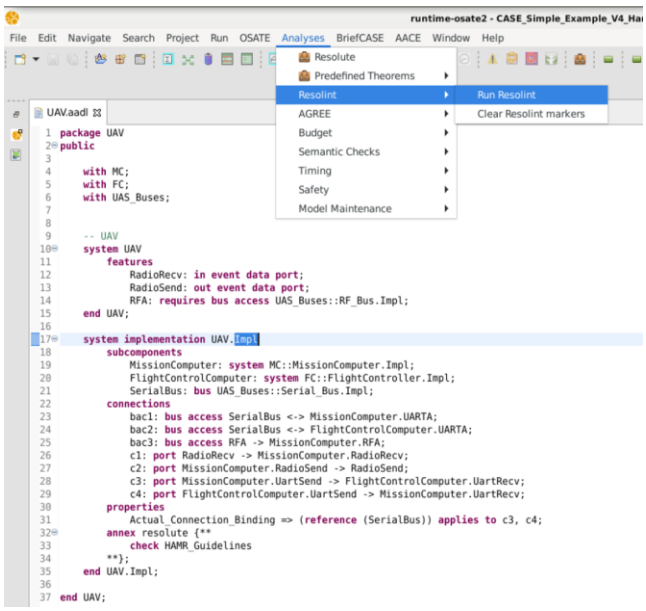
```
system UAS
end UAS;

system implementation UAS.Impl
    subcomponents
        GND: system GS::GroundStation.Impl;
        UAV: system UAV::UAV.Impl;
        RFA: bus UAS_Buses::RF_Bus.Impl;
    connections
        c1: port GND.radio_send -> UAV.radio_recv;
        c2: port UAV.radio_send -> GND.radio_recv;
        bac1: bus access RFA <-> GND.RFA;
        bac2: bus access RFA <-> UAV.RFA;
    properties
        Actual_Connection_Binding => (reference (RFA)) applies to c1, c2;

    annex resolute {**
        -- Make sure this model complies with the CASE toolchain modeling guidelines
        check (CASE_Tools)
    **};

end UAS.Impl;
```

Similarly, individual rules that do not belong to a Ruleset can also be checked, as in the following example:

```
system UAS
end UAS;

system implementation UAS.Impl
    subcomponents
        GND: system GS::GroundStation.Impl;
        UAV: system UAV::UAV.Impl;
        RFA: bus UAS_Buses::RF_Bus.Impl;
    connections
        c1: port GND.radio_send -> UAV.radio_recv;
        c2: port UAV.radio_send -> GND.radio_recv;
        bac1: bus access RFA <-> GND.RFA;
        bac2: bus access RFA <-> UAV.RFA;
    properties
        Actual_Connection_Binding => (reference (RFA)) applies to c1, c2;

    annex resolute {**
        -- Only one processor-bound process can contain thread or thread-group subcomponents
        check ( error( one_process() ) )
    **};

end UAS.Impl;
```

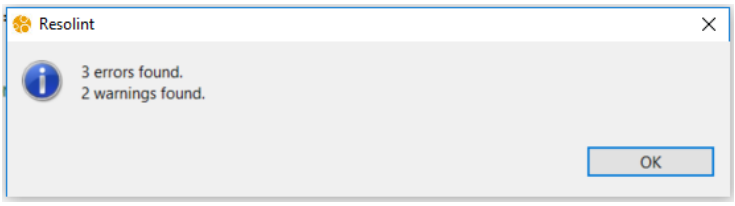
Running Resolint

To run Resolint, select an AADL component implementation containing a *check()* statement in a Resolute annex. From the main menu, select Analyses → Resolint → Run Resolint.

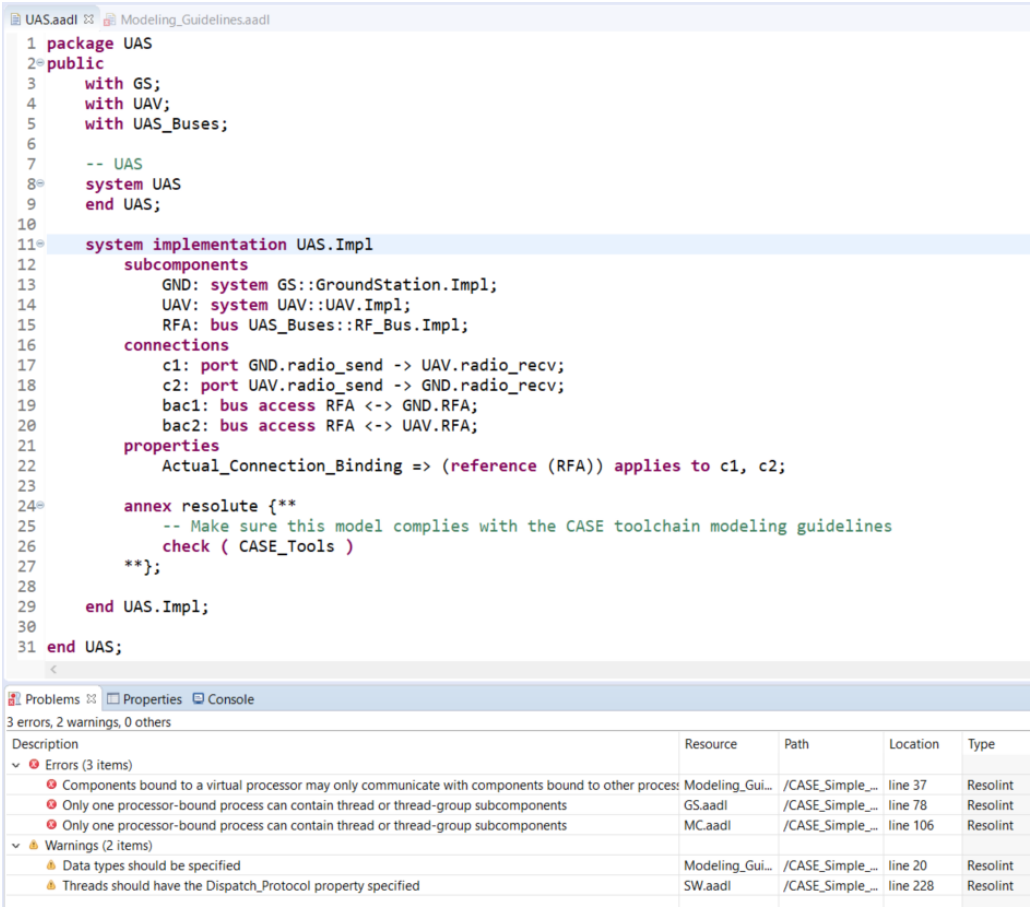


Resolint Output

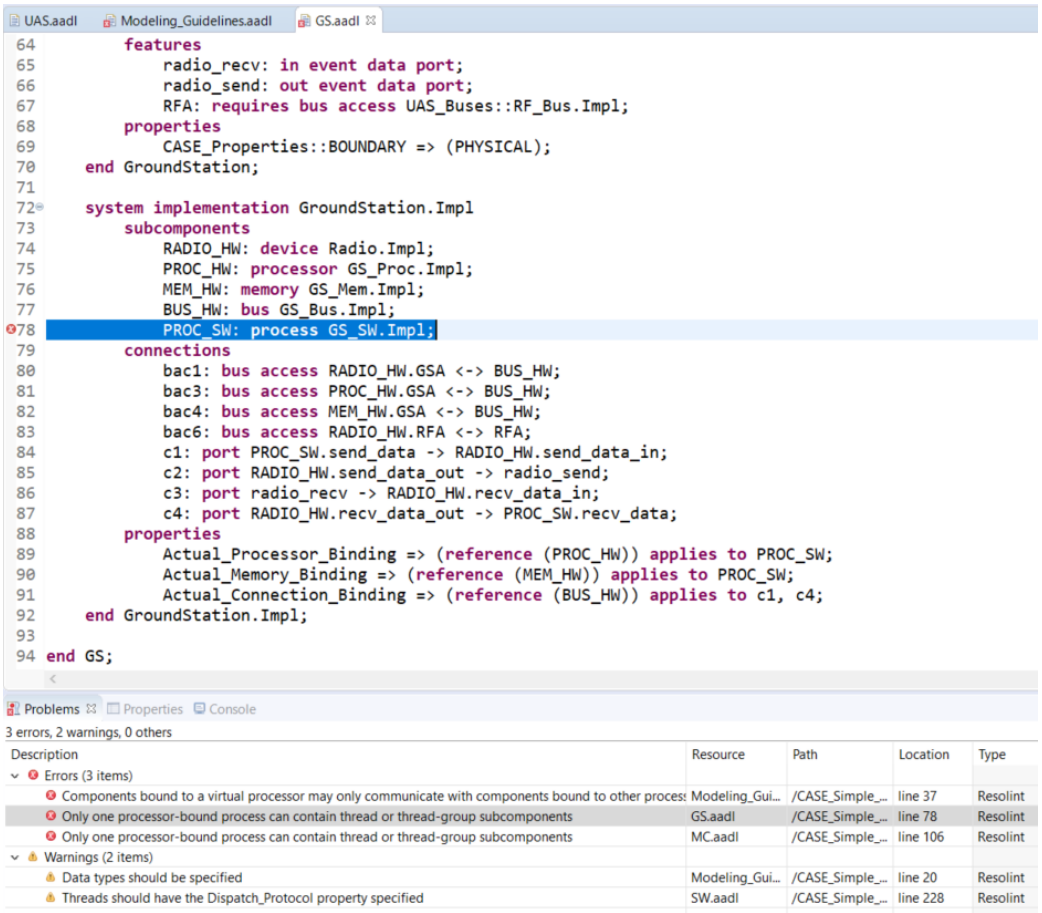
When the Resolint analysis is complete a message box will inform the user whether any rule violations were discovered, and if so, how many of each severity type:



In addition, the list of rule violations will appear in the standard OSATE ‘Problems’ pane:



Double-clicking on an individual problem will open the file containing the AADL element violating the rule and highlight it, as well as place a marker with the corresponding severity in the margin:



Markers can be cleared by either fixing the rule violation and rerunning Resolint, or by selecting Analyses → Resolint → Clear Resolint markers from the menu.