

Error Model Annex: Errata

Peter Feiler, Steve Vestal
phf@sei.cmu.edu, steve.vestal@honeywell.com
Oct 20, 2006

This document contains a set of errata that have been discovered with respect to the Error Model Annex. This annex has been published in June 2006 as SAE document AS5506/1 and is available at <http://www.sae.org/technical/standards/AS5506/1>.

Errata 1: Semi-colon as error property association separator or terminator

Page 100: syntax rules for `error_event_spec`, `error_Propagation_spec`, `error_state_spec`
Page 106: syntax rule for `annex_specific_language_constructs`

For error property associations in the error model type the semi-colon is a separator, i.e., the last declaration does not have one. These error property associations are in curly brackets for each of the error model features. In core AADL and in the error model implementation each property association is terminated by a semi-colon..

For error property associations in error annex subclauses the semi-colon is a separator, i.e., the last declaration does not have one.

Correction P.100:

```
error_event_spec ::=
    defining_error_event_identifier_list : error event
    [ { error_property_association {+ error_property_association }+
    } ];

error_propagation_spec ::=
    defining_error_propagation_identifier_list :
    ( in | out | in out ) error propagation
    [ { error_property_association {+ error_property_association }+
    } ];

error_state_spec ::=
    defining_error_state_identifier_list :
    [ initial [ inactive ] ] error state
    [ { error_property_association {+ error_property_association }+
    } ];

error_property_association ::=
```

```

    ( error_property_identifier
      => error_property_expression
    | nonstandard::error_property_identifier
      => error_property_expression )_);

```

Correction P.106:

```

annex_specific_language_constructs ::=
    annex_error_property_association
{ + annex_error_property_association }+_

```

Errata 2: Numbers in Occurrence property

P. 102 (17): Syntax rule

The occurrence property is defined to take `real_literal` as value. Real-literal is defined in the core AADL as a sequence of one or more digits followed by decimal point and at least one digit after the decimal point. The examples show the values as integers. (Several examples use 10E-4).

Corrections P.102, P.105 (2), P.123(2): Show real numbers in examples.
 10E-4 => 10.0E-4 1E-4 => 1.0E-4

Errata 3: Applies to with component/connection path or list of component features

P. 107 (2) syntax rule

```

annex_error_property_association ::=
    error_property_association
[ applies to [ error ]
  contained_unit_identifier { . contained_unit_identifier }* ]

```

This syntax rule allows a single element (dot-separate path) in the `applies to` clause. This is in accordance with the `applies to` clause of property associations in AADL V1.0.

The `Guard_In`, `Guard_Out`, `Guard_Event`, and `Guard_Transition` examples show a list of component features in the **applies to** clause (for `Guard_Transition` it is a feature or a subcomponent<dot>feature).

```

annex Error_Model {**
  Guard_In =>
    mask when (Sensor_1 and Sensor_2[No_Data])
              or (Sensor_1[No_Data] and Sensor_2)),
  No_Data when (Sensor_1[No_Data] and Sensor_2[No_Data])
              or (Sensor_1 and Sensor_2[Bad_Data])
              or (Sensor_1[Bad_Data] and Sensor_2),
  Bad_Data when Sensor_1[Bad_Data] or Sensor_2[Bad_Data]
applies to Sensor_1, Sensor_2;

```

Corrections P.114, P.124, P.126(3): Show single element in examples by replicating the guard.

```
annex Error_Model {**
  Guard_In =>
    mask when (Sensor_1 and Sensor_2[No_Data])
              or (Sensor_1[No_Data] and Sensor_2)),
  No_Data when (Sensor_1[No_Data] and Sensor_2[No_Data])
              or (Sensor_1 and Sensor_2[Bad_Data])
              or (Sensor_1[Bad_Data] and Sensor_2),
  Bad_Data when Sensor_1[Bad_Data] or Sensor_2[Bad_Data]
  applies to Sensor_1, Sensor_2;
  Guard_In =>
    mask when (Sensor_1 and Sensor_2[No_Data])
    or (Sensor_1[No_Data] and Sensor_2)),
    No_Data when (Sensor_1[No_Data] and Sensor_2[No_Data])
    or (Sensor_1 and Sensor_2[Bad_Data])
    or (Sensor_1[Bad_Data] and Sensor_2),
    Bad_Data when Sensor_1[Bad_Data] or Sensor_2[Bad_Data]
    applies to Sensor_2;
```

Version 2 recommendations:

- Allow lists of elements in **applies** to clause in AADL V2 and Error Model V2
- The optional **error** in the syntax rule is not required

Errata 4: “Port or event feature” – Error model feature or component feature

Annex E.3.4.1 paragraph (13)

It is not clear whether event refers to error model event or the event port. Looks like MetaH terminology where port = data port and event = event port.

Feature refers to component feature, i.e., any port, port group, access, subprogram.

Correction:

The name of a ~~port or event~~ component feature is also permitted in a Boolean error expression that appears in an error state mapping expression. In this case the name refers to the error model associated with the connection made to that ~~port or event~~ component feature.

Errata 5: Derived State Mapping for States or also Propagations?

Annex E.3.4.1 (5) P. 110 Text

*The `Derived_State_Mapping` property defines the mapping from the error states of subcomponents to the error states of the component for which this property is declared. This mapping is only used when the `Model_Hierarchy` property has the value *derived*.*

P.110 Syntax rule:

The syntax rule for `error_source_name` allows naming of propagations (of subcomponents) as well as component features (and their propagation or state).

Correct interpretation: The derived state mapping defines error states as abstract states in terms of error states of subcomponents. Its primary use at this time is to allow the Report property to refer to it instead of the individual states.

Correction: Change syntax rule to be specific to derived state mappings.

```
errorderived_state_source_name ::=
    ( component_name + unique_port_identifier )
    [ [ error_state_or_propagation_identifier_list ] ]
```

(10) A Boolean error expression that appears in an error state mapping expression may contain the names of visible subcomponents, optionally followed by a bracketed list of error state ~~or error propagation~~ names. The error state ~~and propagation~~ names that follow a subcomponent name must be declared within the type of the error model associated with that subcomponent. ~~Propagation names must be declared in the error model as out or in-out propagations.~~ Note this implies that any subcomponents that are named in an error mapping expression must have error models associated with them using the `Model` property.

Errata 6: Inferred Error States

The definition of inferred error state can be found in the semantic description for derived state mapping. It defines the concept of inferred error states of an error propagation.

Correction 1: Move to P. 102 after paragraph (16). This is where the concept of error propagation and error state are introduced.

~~(11) For each **out** or **in out** propagation defined in an error model, the inferred error states for that propagation are the set of error states named as a source state for any transition labeled with that propagation. Note that there may be more than one inferred error state for an error propagation name.~~

Correction 2: Add a sentence about the fact that the Guard_Out declaration extends the set of inferred error states of an outgoing error propagation.

If a Guard_Out is defined for an outgoing error propagation then the set of inferred error states also includes the inferred error states of any incoming features named in the Boolean expression of the Guard_Out.

Errata 7: Guard_In on own propagations?

Annex P.112 E.3.4.2 Text

*The reserved word **self**, when it appears in a Boolean error expression, refers to the component implementation itself. This is used when the value of the expression depends on the error state of the component receiving the error propagation.*

The syntax rule states `error_state_or_propagation_identifier_list` for self.

(3) textually extends the syntax rule for `error_source_name` on P.110. That rule allows naming of subcomponents and features as well as naming of error states and error propagations for both.

Correct interpretation: The intention is to allow naming of incoming features (both ports and data access) and their error propagations and error states, as well as the component's own (self) error states in the boolean error expression of in guards.

Correction 1:

Introduce a revised syntax rule to be specific to in guards.

```
in_guard_source_name ::=
    ( incoming_feature_identifier
      [ [ error_state_or_propagation_identifier_list ] ]
    ) |
    ( self [ [ error_state_identifier_list ] ] )
```

Correction 2:

P.112 (4):

Given an incoming (in or in out) error propagation from each ~~subcomponent~~ access feature or port feature named in a propagation mapping expression, the following rules define how that propagation may be translated into one recognized by the error model of

the receiving component. It is this translated propagation that is used to determine transitions between error states of the receiving component error model.

Correction 3:

P.112 (5):

A ~~component_name or unique_port_identifier~~ (feature name) incoming feature identifier appearing in a Boolean error expression within a propagation mapping expression may be the name of ~~a shared subcomponent~~, a port, or a ~~shared data~~ data/bus/subprogram access feature that is visible within the implementation containing the property association.

Correction 4:

P.112 (6): Delete.

*~~For each component named in a Boolean error expression, any identifiers that appear in an optional bracketed list after the component name must be declared as **out** or **in-out** propagations, or declared as error states, of the error model type for that subcomponent.~~*

Errata 7A: Guard_Out on own propagations?

The same issue holds for Guard_Out (E 3.4.2).

Correct interpretation: The intention is to allow naming of incoming features (both ports and data access) and their error propagations and error states, as well as the component's own (self) error states in the boolean error expression of out guards.

Errata 8: Component features and propagation/state

Annex E.3.4.2 Guard_In paragraph (9) and Guard_Out paragraph (25)
Component should be feature. Propagations should refer to both error states and propagations.

Correction:

*For each feature named in a Boolean error expression, any identifiers that appear in an optional bracketed list after the ~~component~~ feature name must be declared as **out** or **in-out** propagations or declared as error states, of the error model for connections made to that port. If the connection has no error model, then they must be declared as **out** or **in-out** propagations or error states of the ultimate sources of connections made to that port.*

Errata 9: Multiple ports declaration

Annex E.3.4.3 Example

The example declares

```
Sensor_1_Failed, Sensor_2_Failed: out event port;
```

This is not accepted by AADL V1.0. Only single defining identifiers are allowed.

Correction: Split into two declarations

```
Sensor_1_Failed: out event port;
```

```
Sensor_2_Failed: out event port;
```

Errata 10: Conflicting explanation in Error Model Type

Annex E.3.1 paragraphs (26), (27)

(26) gives the impression that Guard_In can only refer to error propagations. Later in the document it is stated that Guard_In can refer to error states as well.

Correction:

There are cases where ~~only the~~ error states and error propagations of an error model type are referenced. For example, the Guard_In property defined in a later section may be used to determine the error state of a component as a function of the errors propagated into it, with or without reference to the error states of the components into which or out of which those errors are propagating.

(27) states that simple identifier matching is used to pair up out propagations and in propagations. Later in the document Guard_In is introduced to support explicit mapping of names. Should put a forward reference here.

Correction (27):

*Distinct error models have distinct name spaces, and names declared in one error model type are never directly referenced in another error model type. The binding of an **out** error propagation to an **in** error propagation is done when the error models of multiple components are composed to form a system error model as explained in a later section. At that time, simple identifier matching is used to pair up an **out** error propagation with an **in** error propagation, or Guard_In and Guard_Out conditions specify a mapping between them.*

Errata 11: Guard_Transitions and “predefined reserved words”

Annex E.3.4.3 (6) P. 116 Text

Not clear what the labels as reserved words refers to.

Correction:

A Guard_Transition property association declares any voting or consensus protocols used for ~~in~~ incoming port events that appear in mode transition declarations.

The property expression resembles an error state or error propagation mapping expression, except that the labels on the rules are predefined reserved words that specify alternative mode transition behaviors when errors propagate through the event connections that trigger the mode transitions.

Errata 12: optional applies to on Error Property Expressions

P. 107 (2) syntax rule

```
annex_error_property_association ::=
    error_property_association
    [ applies to [ error ]
      contained_unit_identifier { . contained_unit_identifier }* ]
```

indicates that any error property association in an annex subclause has an optional applies to statement.

For the Model property, the Report property the applies to is optional. For all of the guards the applies to is required.

Correction:

In each of the guard sections add a statement that applies to is required.

Future correction:

Change the grammar rules to distinguish between the different types of error property associations.

Errata 13: No Subcomponent references in Guard_In or Guard_Out rules

Annex E.3.4.2 Paragraphs (6) and (22), (24)

These paragraphs talk about the guard rule expression referring to subcomponents. They should be deleted.

The reason: In_Guard and Out_Guard expressions are defined in terms of self error state and error propagations or error states of incoming features.

(4) (22) Given an error propagation from each ~~subcomponent~~ or feature named in a propagation mapping expression, the following rules define how that propagation may be translated into one recognized by the error model of the receiving component. It is this translated propagation that is used to determine transitions between error states of the receiving component error model.

*(6)(24) For each component named in a Boolean error expression, any identifiers that appear in an optional bracketed list after the component name must be declared as **out** or*

~~in-out propagations, or declared as error states, of the error model type for that subcomponent.~~

Errata 14: Guard_Out Rule Mapping

Annex E.3.5.5 Paragraph (3) states that Guard_Out maps out propagations into out propagation identifiers.

Correction: It maps error propagations or error states of incoming features and its own error states into an outgoing error propagation of the given component.

(19) A Guard_Out property specifies how ~~out~~ error propagations or error states from incoming features and error states of a component are translated or masked before being propagated out of that component. Guard_Out property associations may be declared for shared subcomponent, port, data, and client and server subprogram features of a component and apply to error propagations that occur through those features, for example through an out data port or a shared data object.

Errata 14 A: Guard_Out rules defined for Out Propagations

Annex E.3.4.2 Paragraph (29) suggests that the label preceding the **when** refer to an **in** propagation of another component; in other words the Guard_Out specification in one component would specify a condition on a propagation in another component. Paragraph (29) should be changed to refer to **out** propagation of the component for which the guard is defined.

Correction: E 3.4.2 (29)

*Within an error propagation rule in a Guard_Out property expression associated with a shared subcomponent, each propagation identifier preceding a **when** must be the name of an **in** or **in-out** propagation declared in the type of the error model associated with the shared subcomponent. Within an error propagation rule in a Guard_Out property expression associated with an **out** or **in-out** feature, each propagation identifier preceding a **when** must be the name of an **in-out** or **in-out** propagation declared in the type of the error model associated with connections made to that feature. If any connection does not have an associated error model, then each propagation identifier preceding a **when** must be the name of an **in-out** or **in-out** propagation declared in the type of the error model associated with ultimate source for that connection.*

Errata 15: Guard_Event rule clarification

Guard_Event specifies conditions under which port event are triggered through event ports or the error event data port. Explicitly state what can be included in the condition.

Correction:

*(1) A `Guard_Event` property association declares that an event is raised when a particular error condition is detected by a component as a result of voting or consensus protocols performed by the code or circuitry of that component. The object identifier for which this association is defined must name a visible **out** event feature or the error event data-port feature of the component containing the association. The associated expression must be a single Boolean propagation expression. This Boolean expression specifies the condition for raising the port event in terms of error propagations or error states of incoming features and in terms of error states of the component itself (self).*

Errata 16: `Guard_Transition` rule definition

Annex E.3.4.3

Paragraphs (6),(7) & (8) should talk about in ports of the component and out ports of subcomponents, i.e., any event port that can trigger a mode transition.

(6) A `Guard_Transition` property association declares any voting or consensus protocols used for ~~in~~ events that appear in mode transition declarations. These events come from outgoing event ports of subcomponents or incoming event ports of the component itself. The property expression resembles an error state or error propagation mapping expression, except that the labels on the rules are predefined reserved words that specify alternative mode transition behaviors when errors propagate through the event connections that trigger the mode transitions.

(7) If any event arrives for an event port named in a mode transition in the scenario being used for error modeling and analysis, then a `Guard_Transition` property association must be declared for the ~~in~~-event port named in that mode transition.

(8) The object identifier in the property association must be the declared name of an ~~in~~ event port, where the name of that event port appears in at least one mode transition declared within the component implementation that contains the property association.

Errata 18: Error Models and Connections

The Annex permits error models to be associated with connections. This can be done through the **applies to** clause on the MODEL property referring to a connection declaration.

A statement should be added to indicate that `Guard_In`, `Guard_Out`, `Guard_Event`, and `Guard_Transition` do not apply to error models on connections.