AADL EMV2 Errata

Peter Feiler

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

DM18-0659

Version Management of EMV2

Implementation of approved errata

New name: EMV2_1

- BA did not get a new name

Compatibility with EMV2

Timing of new version release

Errata

Issue #19: Name path consistency

Issue #39: Error type library naming

Issue #40: User defined propagation paths to reference features

Issue #35: Error propagation declarations and references

Multiple issue entries: Trigger condition logic

Issue #8: Configure initial state

Issue #12: EMV2 does not deal with reference to array elements for subcomponents and features: k of n logic handles arrays. Need for specifying individual elements

Issue #41: Propagation paths within address space

Name Path Consistency

We have syntax for applies to in properties (or reference values)

- In core: a.b.c@(emv2)es.ep
- In EMV2: “escape” to core: ^sys.a.b.c@ep.et escape character and @ without annex name
- In EMV2: es.ep.et#Occurrence => 2.5;
- et is type thus should use {et} as we do for propagation constraints es.ep{et}#Occurrence => 2.5;
- ^sys.a.b.c@ep.et{et}#Occurrence => 2.5;

In core element references in EMV2 constructs

- Subcomponents & state in composite: s1.s2.estate1 => ^s1.s2@estate1
- subcomponent out propagations in out propagation conditions: ^s1@ep1

- Compatibility: We need escape character ^ (cannot be optional)
- Documentation: Will be part of V3 core
- handle the same way we deal with mode specific values

Error Type Library Name

Currently error type library inherits the name of the package

- {** **error types** … **end types;**

Proposal:

- Provide explicit name for type library
- Allows for multiple type libraries in same error library declaration
- Either unnamed in package, or named only in package, not both in same package
- Question of fully qualified name if reference between error type libraries in the same package

- Not now
- Discussion: name spaces and annex libraries (V3)

User Defined Propagation Paths

Currently the path declaration limits the end points to user defined propagation points

```
propagation_path ::=

    defining_observable_propagation_path_identifier :
     source_qualified_propagation_point ->
       target_qualified_propagation_point ;


qualified_propagation_point ::=
    { subcomponent_identifier . }* propagation_point_identifier
```

Use case: add propagation point to outer component and define a path from an out feature of a subcomponent

Need to look at how we describe features vs. error propagations to be propagation points (features that have error propagation declaration).

Proposal:

- Allow reference to features as well

Error Propagation Declaration

Declaration and naming

- featurereference: (**in** | **out**) **propagation {** <errortypes> **}**
- featurereference**: not (in | out) propagation {** <et> **}**

For in out features we have an in propagation and an out propagation

In addition we have a containment for in and out

When referencing an error propagation, e.g., to assign a property value how do we distinguish between each of the four instances?

Options:

- Combine propagation and containment into a single declaration
  - Featurereference: **in propagation {** <et> **} not {** <et> **};**
  - A type cannot be in both propagation and containment thus we have a unique type reference
- Explicit name for each propagation to distinguish in from out for in/out features

Compatibility: requires models to be converted (do in next major version)

Logical expression issues

Binary exclusive or Operators

- Keyword OR: XOR semantics
- A1 XOR A2 XOR A3
  - Boolean logic interpretation: A1 = T A2 = T A3 = T => T
  - Intended interpretation: XOR (A1, A2, A3), i.e., one failure only

Event and state based condition evaluation

- Error events are occurring independently
  - Only one at a time can trigger a transition
  - Should not directly impact outgoing propagation condition
- Error propagations reflect error state
  - Evaluation of multiple is possible

Logical operators on propagations

Existing operators

- k **ormore** n: inclusive or starting with k subset
- k **orless** n: inclusive or up to k subset

Proposal: add k **of** n operator

- Any subset of size k failure
- 1 **of** (a1, a2, a3) exactly one failure
- Use case: triple redundant sensors
  - If one fails go to degraded mode
  - If 2 or more fail go to failstop mode

Support arrays

- 1 **ormore** (p1, p2, p3)
- 1 **ormore** (p[3])

Binding points: potentially unknown # of propagation sources

- Does conditional logic make sense on bindings

Principals of Fault Logic Operators

Operators on collection of fault triggers

- We already have k ormore(T)
  - T = collection of triggers
  - n = size(T)
- Related to the count of how many trigger conditions hold
  - count(T)
- count(T) = n : **All**(T)
  - same as multi-operand **and**
- count(T) = k : **k of**(T)
  - 1 of(T) same as multi-operand exclusive or
  - Often equated with k of n voting
- k <= count(T) <= n : k **ormore**(T)
  - 1 ormore(T) same as inclusive or
  - Sometimes equated with k of n voting
- 0 <= count(T) <= k : k **orless**(T)
- j <= count(T) <= k: j..k **of**(T)

Practical Set of Operators

Minimal set of operators

- All, 1 of
- Others can be expressed in terms of those two

Nested operators: expressions as elements of T

- Restricted to a normal form
- Full set

Context specific restrictions

- Composite state expressions
  - States only
- Transitions
  - Meaning of multiple transitions between same states (“or” or “xor”)
  - No “and” of multiple events (“xor” since events are not simultaneous?)
- Out propagation conditions
  - Implicit state: state on left of statement
  - Incoming propagations only (incl. subcomponent out prop)
- Detection conditions
  - State, incoming propagations

Configure Initial Error State

Currently the initial state is indicated by a keyword

There are use cases where we may want to specify that at different times there is a different initial state, e.g., a component may start out in a failed state.

“initial” keyword is misleading. Maybe nominal or default. *Stay with “initial”*

Proposal:

- Allow specification of starting state via property
  - Tag the state with a Boolean Starting_State property
    - (would need rule that only one state per SM can have the property)
  - Per state machine a property identifies starting state by name (reference) (OK)
    - Should not be mode specific (OK)
    - Need to be able to associate property with “subclause” (state machine instance)
- Specify a configuration of starting states for a system
  - Collection of property associations, one for each component whose error state needs to differ from the default initial state

Propagation Paths
From EMV2 Standard document

a process, thread group, or thread to every other process, thread group, or thread that is bound to any common virtual processor, processor or memory (Note that address space boundary is enforced at the process level, i.e., two threads inside the same process may affect each other beyond the specified error propagation points).

How to model propagations across threads inside an address space (process)?

- Via binding to virtual processor or processor
- Implicit or explicit at thread/process level
  - Naming of propagation point
  - Specification of error types being propagated

Type Products

Clarification of their role and use

- Errata about not being able to associate properties with type product

Logical Expressions in Error Flows

Effectively merge out propagation conditions and error path specifications