AADL EMV2 Errata Sept 2017

Peter Feiler

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM17-0701

Errata

Issue #11/#26: out propagation condition in terms of subcomponent out propagation: Approved

Issue #28: typeset on right of transition or out propagation conditions should be a single type: Approved

Issue #29: Allow state declarations in component error behavior: “Can be done through library so no change”

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

Revision

Trigger condition logic

Name path consistency

Version management of EMV2
Implementation of approved errata

Propagation paths within address space

Error propagation declarations and references

Error type library naming

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

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: et is type thus should use {et} as we do for propagation constraints ep{et}

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

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

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

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:

- Explicit name for each propagation to distinguish in from out for in/out features
- 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