AADL Annex Syntax Revision

Peter Feiler

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Revision Objectives

Reduce syntactic noise

- Annex emv2 {** **} => @e{ }

Closer integration with core language

- Allow annex annotations to model elements
  - Example: allow error types on features
    with errorlibrary; p1: in port dt @e{ types } ;
    instead of
    p1: in port dt; annex emv2 {** use types ErrorLibrary; feature in propagation { types }; **};

Ability to configure in annex annotations

Annex library without language embedding

- One annex library per package – can be processed directly by sublanguage compiler

Utilize type system and properties

- Error types expressed by type system
- Properties on annex model elements
- Type specific property values: similar to mode specific values, other mapped property values

Can we fold EM/Behavior spec into core more transparently?

Configuration/composition of Annex Subclauses

Adding in annex specifications

- Annex subclauses may be declared in a separate classifier extensions
- Different annex specifications may be added

```
System Top_emv2 extends top is
Annex EMV2 {**
  use types ErrorLibrary;
  …
**};
End Top_emv2;
```

```
Configuration Top_emv2 extends top
@e { use types ErrorLibrary; };
End Top_emv2;
```

Use @ consistent with reference path usage

Use {} instead of {** **} with {} usage restriction within annex
Alternative option: use another symbol pair e.g. <>

```
Configuration Top.config_full extends Top.config_L2, Top.flows, Top_emv2 end;
```

Inherited annex subclauses based on classifier **extends**

- Automatically included
- Extends override rules of annex apply

Separate extensions

- No conflicting declarations

Assignment of Annex Annotations to All Instances

Configurations as annotations for all subcomponents of a given classifier

- Example: EMV2 configuration for a classifier
- Assign annotations individually

```
Configuration AvionicsSystem.Dual is
  FlightSystem2 => FlightSystem.primary;
  all(Mine::Sensor) => Sensor.emv2;
  all(Mine::GPS) => { @e { … };};
```

```
Configuration Sensor.emv2 extends
Sensor
@e { use types ErrorLibrary; };
End Top_emv2;
```

- Specify collection of EM annotations

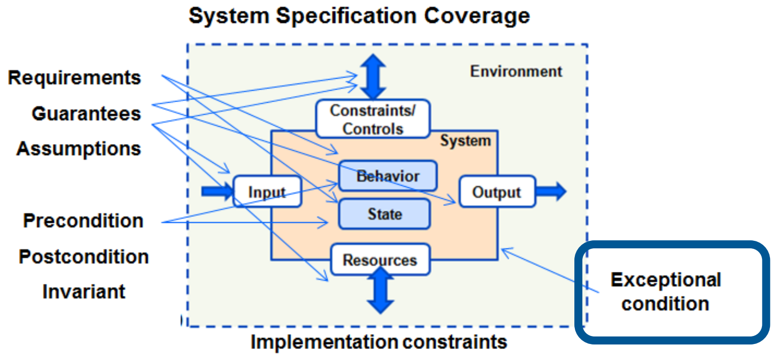
Simplified syntax does not require @e{} inside {}

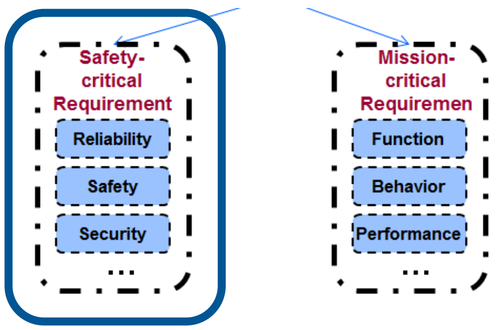
```
Configuration FlightSystemEMV2 is
  all(Mine::Sensor) => Sensor.emv2;
  all(Mine::GPS) => @e { … };
end;
```

- Assign configuration pattern to subsystems

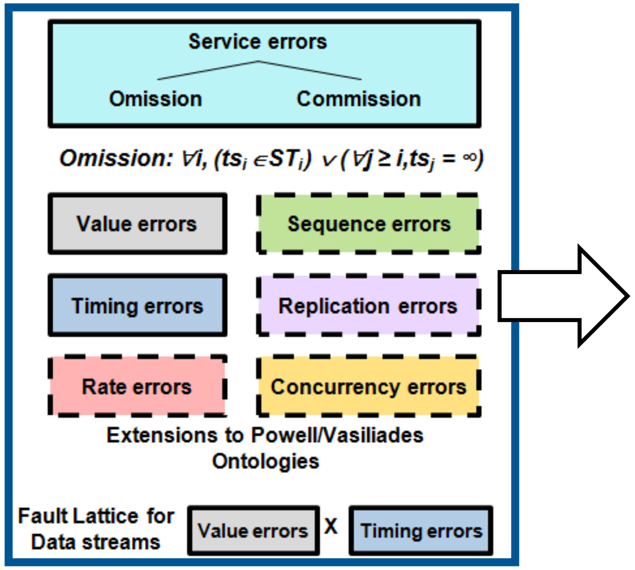
```
Configuration AvionicsSystem.Dual is
  FlightSystem2 => FlightSystem.primary, FlightSystemEMV2 ;
```

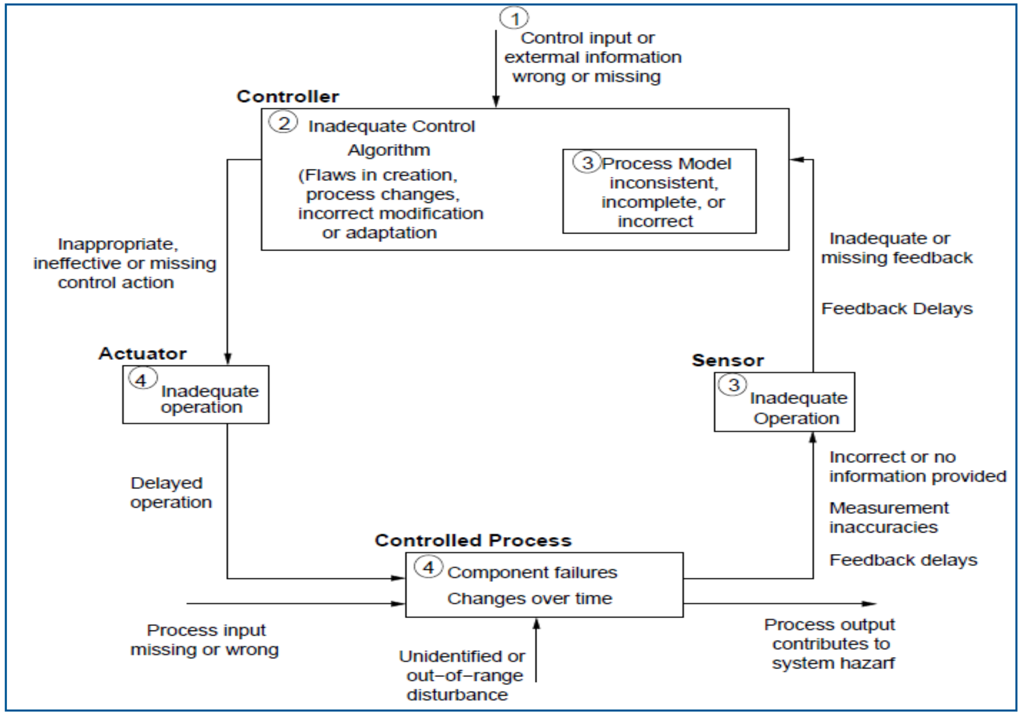
Extension of Architecture-led Requirement Specification





Error Propagation Ontology




1 Control input or external information wrong or missing
Controller
2 Inadequate Control Algorithm (Flaws in creation, process changes, incorrect modification or adaptation)
3 Process Model inconsistent, incomplete, or incorrect
Inappropriate, ineffective or missing control action
Inadequate or missing feedback
Feedback Delays
Actuator
4 Inadequate operation
Sensor
3 Inadequate Operation
Delayed operation
Incorrect or no information provided
Measurement inaccuracies
Feedback delays
Controlled Process
4 Component failures
Changes over time
Process input missing or wrong
Unidentified or out-of-range disturbance
Process output contributes to system hazarf

Integration of EM/Behavior into Core

Example: error propagation

- Error types on features
- @e syntax vs. specific syntax

Flows and IPO processing logic

- All I – All O (no flow spec)
- Pairwise flow (path, source, sink in V2/EMV2)
- Multiple I to O
  - EMV2: s –[i1 and i2 or i3]-> o1
- I – function – O
  - Subprogram entrypoint with parameter connections including mode specific
  - BA: action language with mode specific behavior.
  - EMV2: component internal error/recover events: failure mode specific behavior

State behavior

- State machine instance for modes, error behavior, application control flow, monitored external system state

EM Integration into Core

Error types as types in type system

- Was in EM library

Error behavior state machine as generic reusable SM definition

- Was in EM library

Error propagation: type set (union?) on features

Error flow: flow with type set as pre and post condition

Out propagation condition: I to O mapping with AND/OR/k OF n logic

Error/recover event: similar to internal events in core

- Recover event: proxy for core action such as mode transition

Transition condition: state transition with trigger conditions base on events/I with type set filter

Detection condition: “core” function (via internal event) based on EM (state, I typeset conditions)