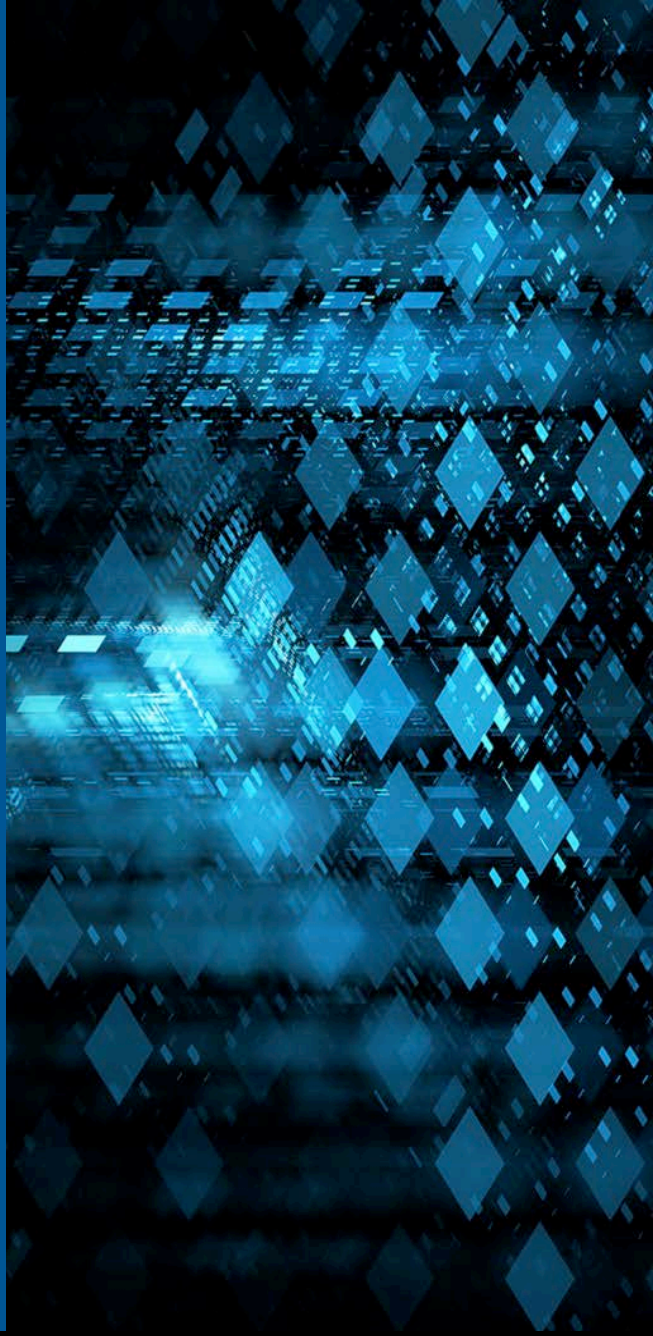
AADL for Secure & Safe Systems Design & Analysis

Part 4 - Safety

Julien Delange

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. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM-0003990

Tutorial Agenda

Introduction: required background, role of MBE, tutorial overview

AADL Concepts: learn enough to use AADL and OSATE

Flow Latency: how to capture flow characteristics? How can I generate a flow analysis from my architecture model?

Safety Analysis: how to capture safety in an AADL model? What types of reports can I generate? How can I generate them?

Security Analysis: representation of security aspects. How to detect security issues? What type of reports can we generate?

What is the goal of Error Model Annex?

Capture safety hazards: what hazard can happen in my system and how they can propagate through architecture elements – features, subcomponents, etc.

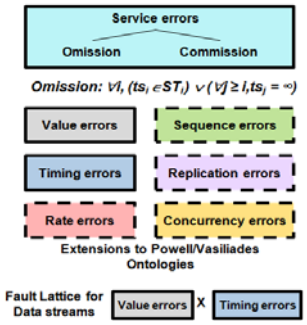
Analyze potential unforeseen safety issues: how hazards propagate and impact other components.

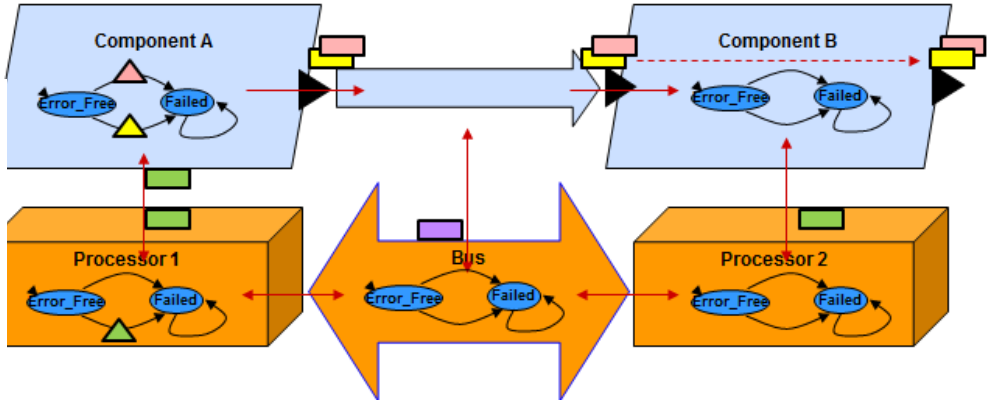
Automate the safety evaluation process: safety analysis is long and labor-intensive (e.g. ARP4761). Require multiple reports: FMEA, FTA, FHA, etc. Automating will help to keep certification documents up to date and help to detect unexpected safety issues.

Integration in the same architecture model: no multiple, inconsistent models – one, single representation – an AADL model - for driving several analysis – latency, safety, security.

Error Model V2: Four Levels of Abstractions

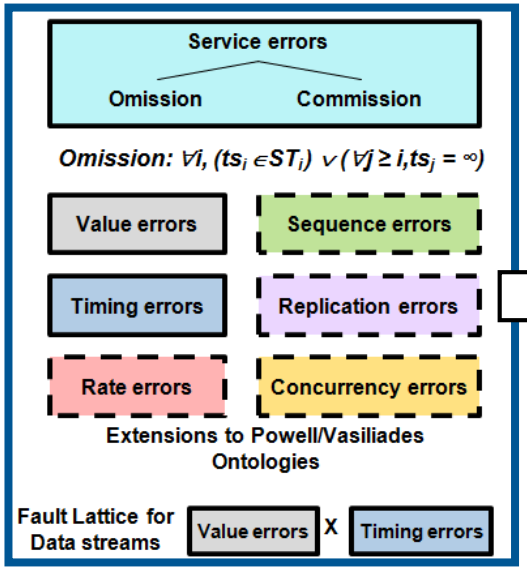
- **Focus on fault interaction with other components**
  - Probabilistic error sources, sinks, paths and transformations
- **Focus on fault behavior of components**
  - Probabilistic typed error events, error states, propagations
  - Voting logic, error detection, recovery, repair
- **Focus on fault behavior in terms of subcomponent fault behaviors**
  - Composite error behavior state logic maps states of parts into (abstracted) states of composite
- **Types of malfunctions and propagations**
  - Common fault taxonomy
  - User definable types

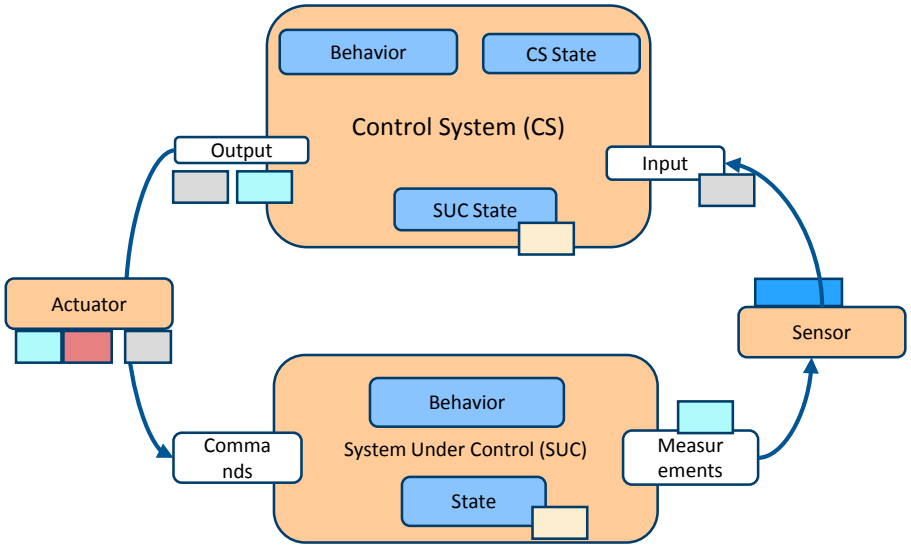




Fault Taxonomy Guides Error Model EMV2 Annotations

Error Propagation Ontology &
Error Model V2 Annex






```
error propagations
    SourceTracks:out propagation{SensorDataOmission,SensorDataOutofRange};
    flows
        src1:error source SourceTracks{SensorDataOmission} when Failed;
        src2:error source SourceTracks{SensorDataOutofRange} when Degraded;
end propagations;
```

Error Sources
Omission/Commission
Early/late
Out of range/incorrect value
Wrong rate/duration
Inconsistent SUC state

Error Type Libraries

Error Type libraries and AADL Packages

- An AADL package can contain one Error Model library declaration
- The **error types** clause represents the Error Type library within the Error Model library
- The Error Type library is identified and referenced by the package name

Error Type library represents a namespace for error types and type sets

- Error type and type set names must be unique within an Error Type library
- An Error Type library can contain multiple error type hierarchies

```
Package myerrortypes
public
Annex emv2{**
error types
    AxleFailure: type;
    Fracture: type extends axlefailure;
    Fatigue: type extends axlefailure;
end types;
**};
End myerrortypes;
```

Error Types & Error Type Sets

Error type declarations

```
TimingError: type ;
EarlyValue: type extends TimingError;
LateLate: type extends TimingError;
ValueError: type ;
BadValue: type extends ValueError;
```

Error Type Set as Constraint
{T1} tokens of one type hierarchy
{T1, T2} tokens of one of two error type hierarchies
{T1*T2} type product (one error type from each error type hierarchy)
{NoError} represents the empty set
Constraint on state, propagation, flow, transition condition, detection condition, outgoing propagation condition, composite state condition

An *error type set* represents a set of type instances

- Elements in a type set are mutually exclusive
- An error type with subtypes includes instances of any subtype
- A *type product* represents a simultaneously occurring types
  - Combinations of subtypes

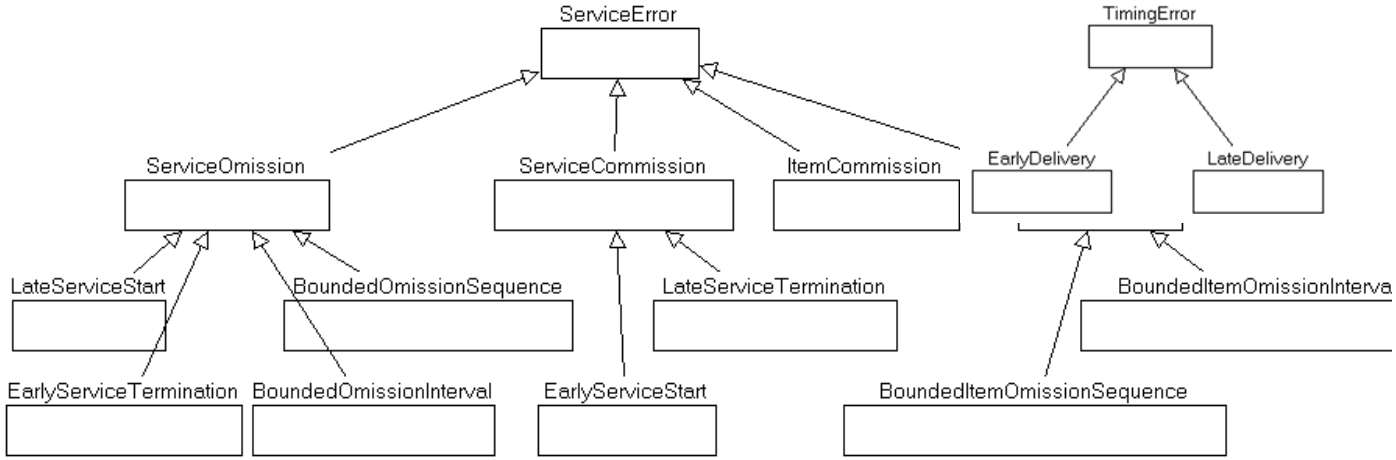
```
InputOutputError : type set {TimingError, ValueError, TimingError*ValueError};
```

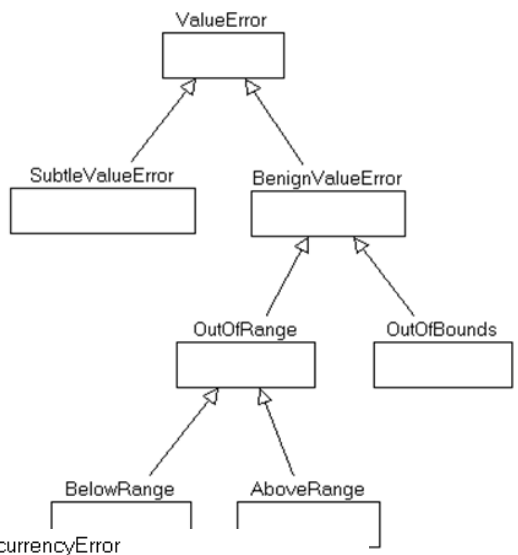
An *error type instance*

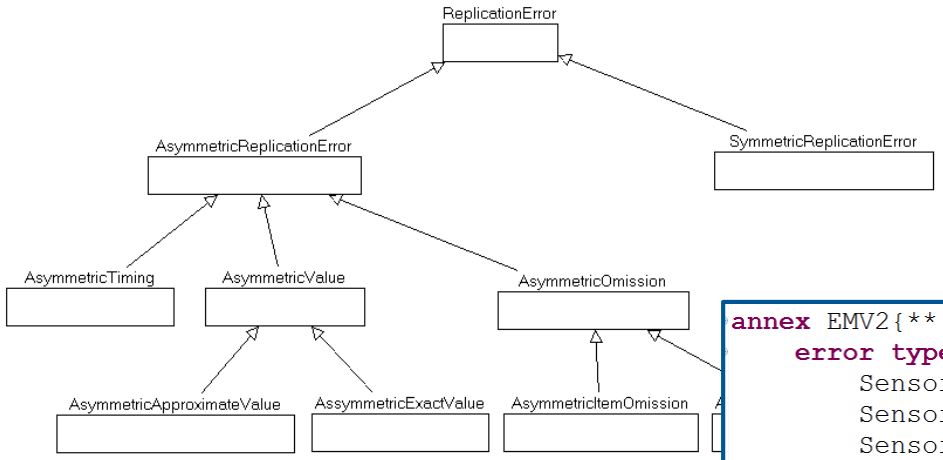
- Represents the error type of an actual event, propagation, or state
- Like a token in a colored Petri net

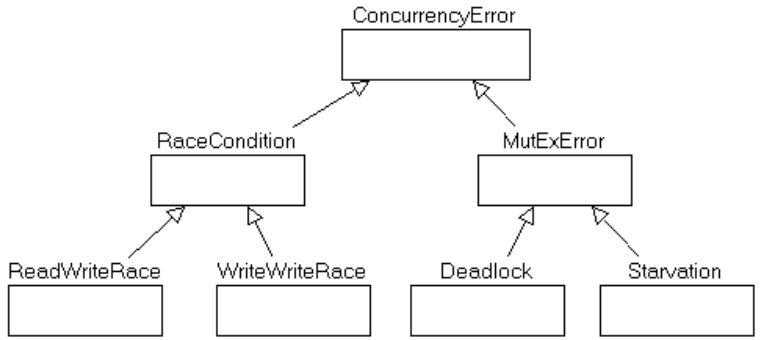
```
{LateValue * BadValue}
{LateValue}
```

Error types taxonomy







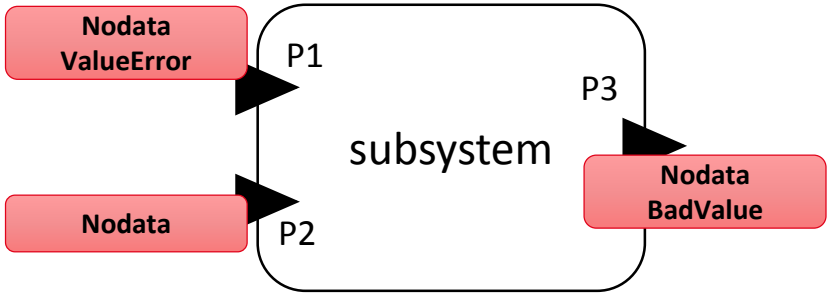



```
annex EMV2{**
    error types
        SensorDataOmission:type extends ErrorLibrary::ItemOmission;
        SensorDataOutofRange: type extends ErrorLibrary::OutofRange;
        SensorDataClippedHigh: type extends ErrorLibrary::OutofBounds;
        SensorDataClippedLow: type extends ErrorLibrary::OutofBounds;
        SensorDataBiasedHigh: type extends ErrorLibrary::SubtleValueError
        SensorDataBiasedLow: type extends ErrorLibrary::SubtleValueError;
        SensorDataDrifting: type extends ErrorLibrary::SubtleValueError;
```

Error Propagation Declarations

Core language

```
system subsystem
features
    P1: in data port;
    P2: in data port;
    P3: out data port;
```



Error Model Annex

```
annex EMV2 {**
    use types ErrorLibrary;
    error propagations
      P1: in propagation {NoData, ValueError};
      P2: in propagation {NoData};
      P3: out propagation {NoData, BadValue};
end propagations;
**};
end subsystem;
```

The component receives `Nodata` and `ValueError` on `P1`, `NoData` on `P2` and `NoData` and `BadValue` on `P3`

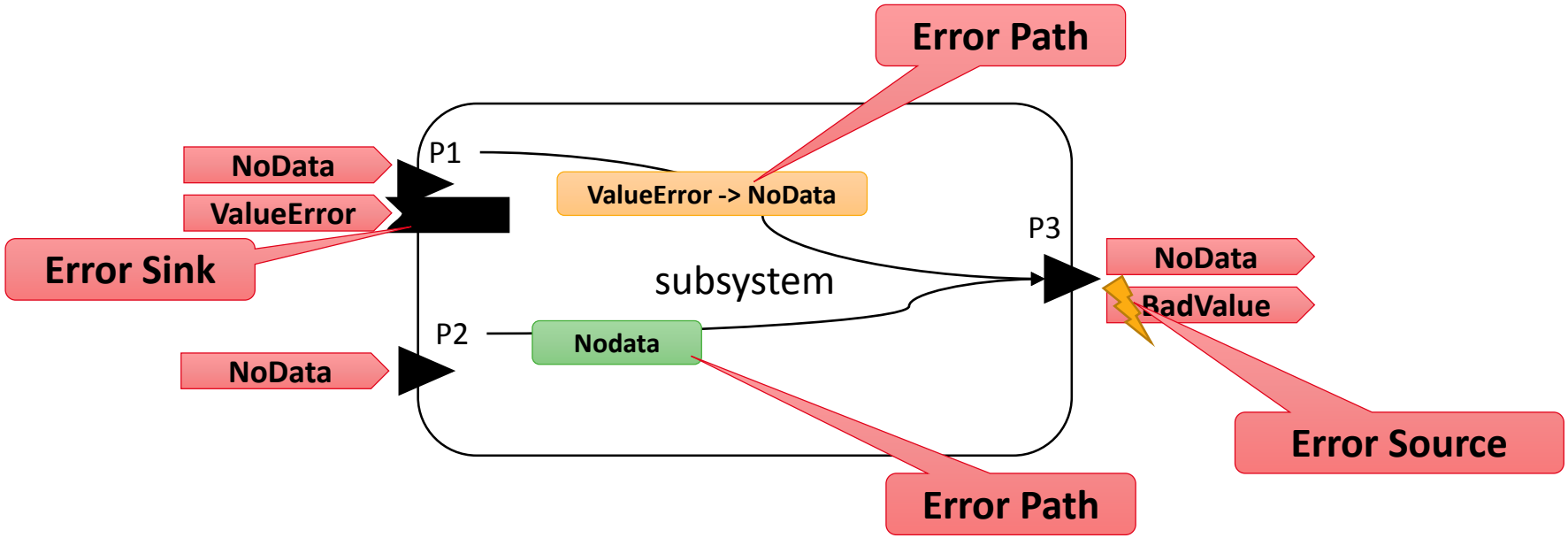
Error Flows

Error flow specifies **the role** of a component in **error propagation**

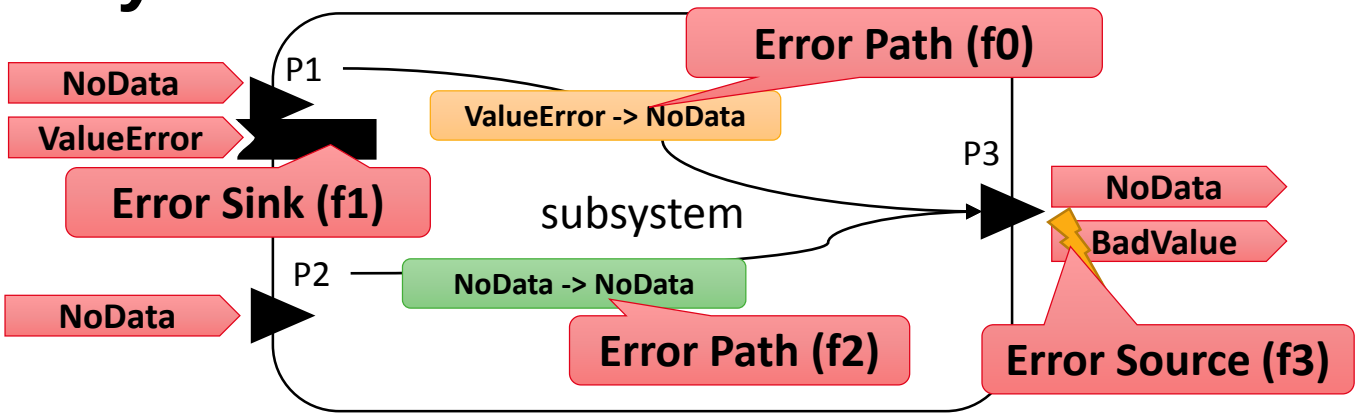
- The component may be a **source or sink** of a propagated error types
- The component may pass **incoming types through as outgoing types**
- The component may **transform an incoming type into a different outgoing type**
- By default all incoming errors of any feature flow to all outgoing features

Used for **Functional Hazard Assessment**

Use for **Fault Impact Analysis**

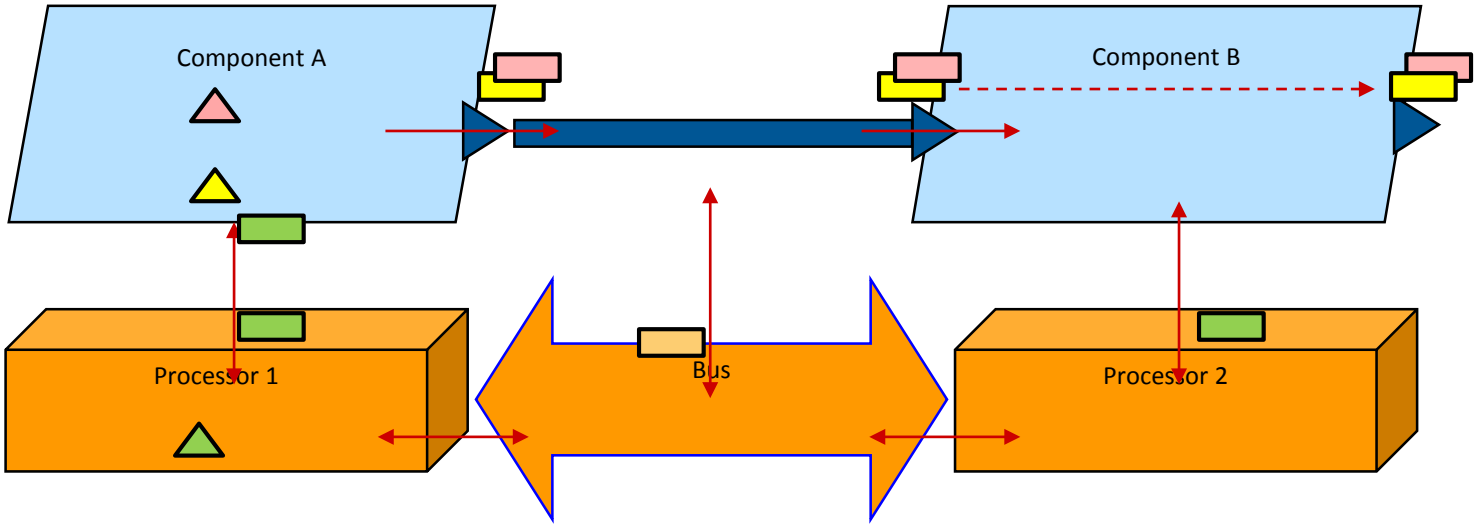


Error Flows – syntax definition




```
system subsystem
features
    P1: in data port;
    P2: in data port;
    P3: out data port;
annex EMV2 {**
    use types ErrorLibrary;
    error propagations
      P1: in propagation {NoData, ValueError};
      P2: in propagation {NoData};
      P3: out propagation {NoData, BadValue};
    end propagations;
    flows
      f0 : error path p1{ValueError} -> p3{NoData};
      f1 : error sink p1{NoData};
      f2 : error path p2{NoData} -> p3{NoData};
      f3 : error source p3{BadValue};
    end flows;
**};
end subsystem;
```

Propagation Paths are Determined by The Architecture



a processor to every thread bound to that processor and vice versa

a processor to every virtual processor bound to that processor and vice versa

a processor to every connection bound to that processor and vice versa

...

a virtual bus to every connection bound to that virtual bus and vice versa

a device to every connection bound to that device and vice versa

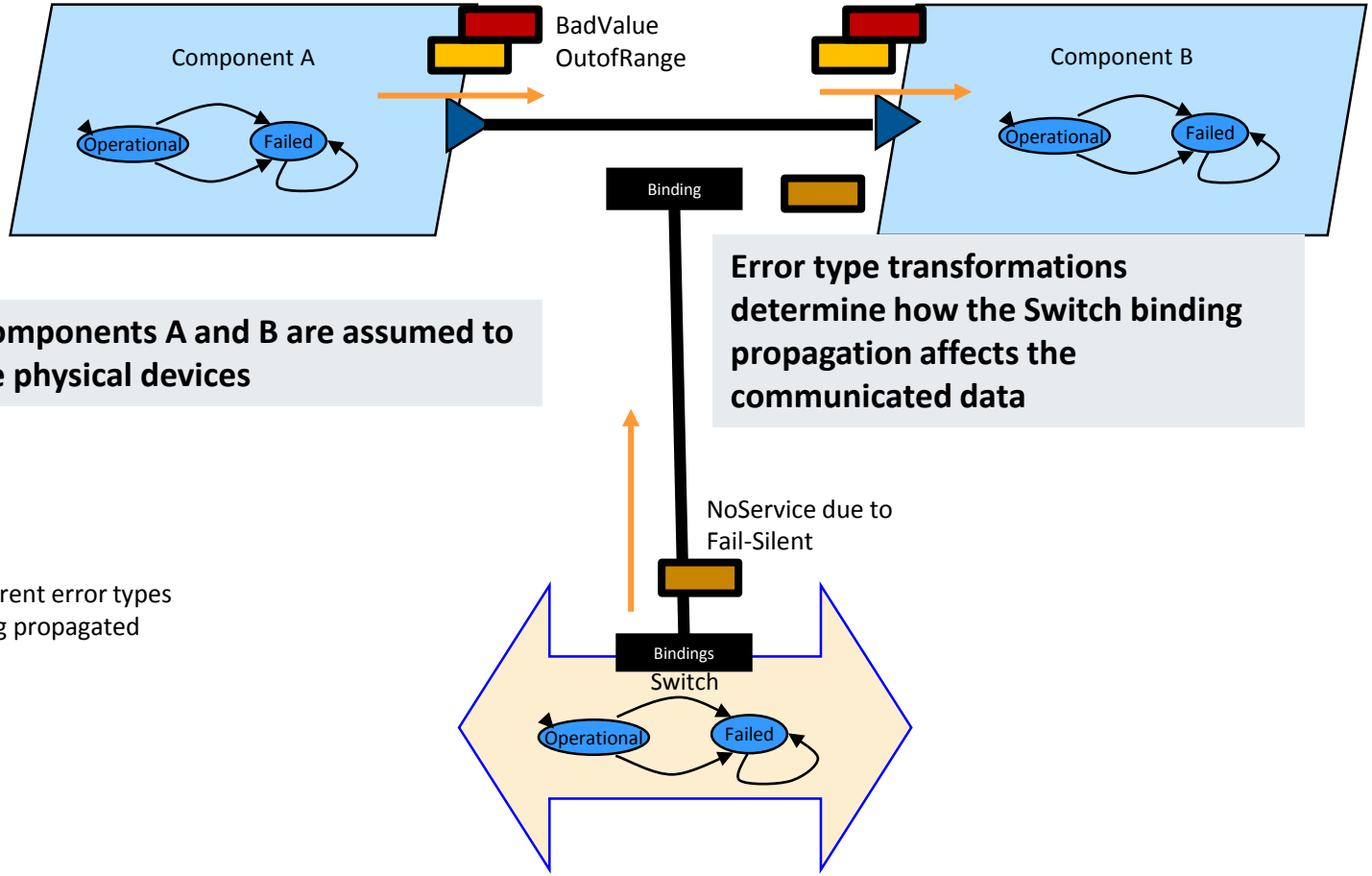
a component to each component it has an access connection to and vice versa, subject to read/write restrictions

a component from any of its outgoing features through every connection to components having an incoming feature to which it connects

...

Connection Binding Error Propagation

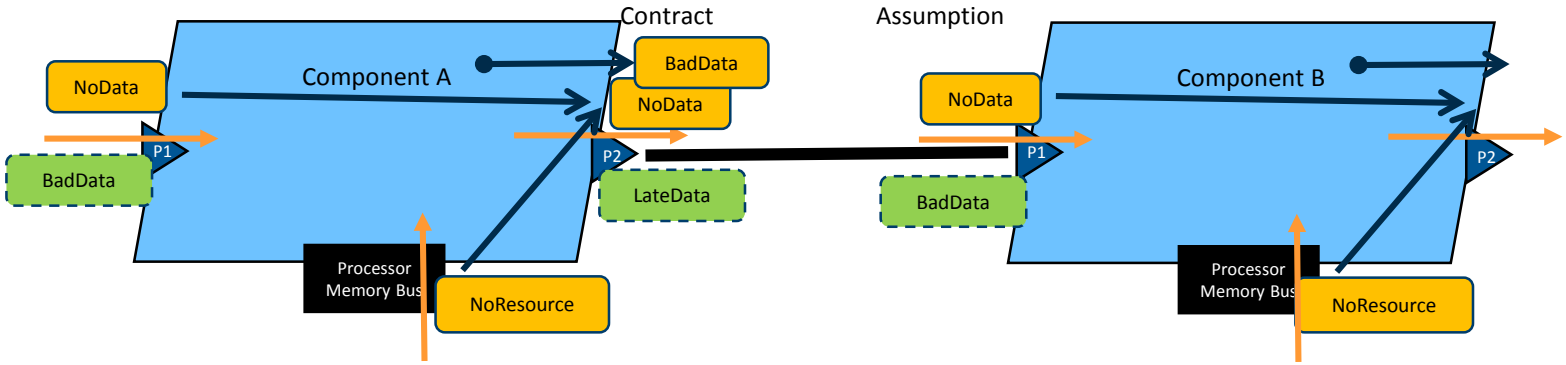
Error flow is determined by data flow between components, e.g., along port connection



Components A and B are assumed to be physical devices

Error type transformations determine how the Switch binding propagation affects the communicated data

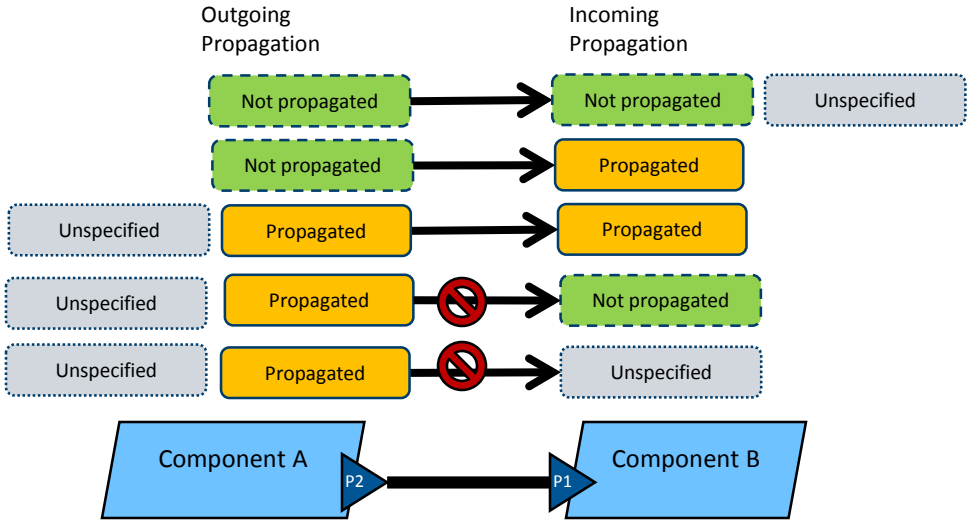
Consistency in Error Propagation



Present and absent outgoing and incoming error propagations

Error sources, paths, and sinks (FPTC)

Connections as error sources



Mismatched fault propagation and containment assumptions

Discovery of unhandled error propagations.

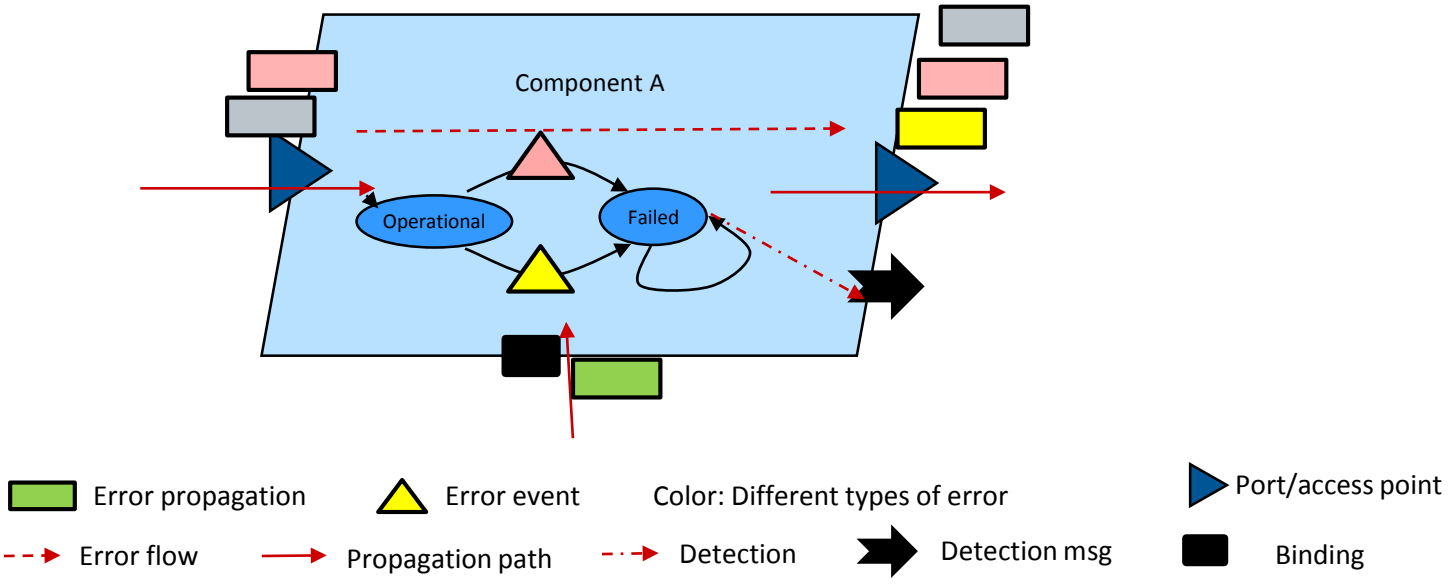
Component Error Behavior

Components have error, mitigation, and recovery behavior specified by an error behavior state machine

Transitions between *states* triggered by *error events* and *incoming propagations*.

Conditions for *outgoing propagations* are specified in terms of the *current state* and *incoming propagations*.

Detection of error states and incoming propagations is mapped into a message (event data) with error code in the system architecture model



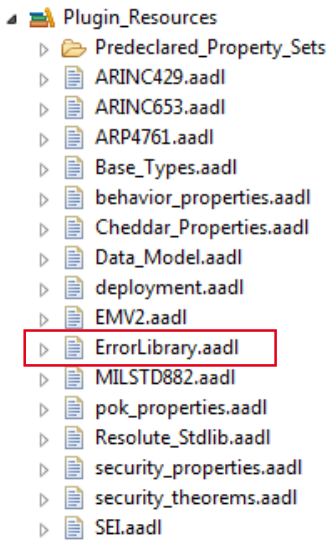
Reusable Error Behavior State Machine

Declared in a package context

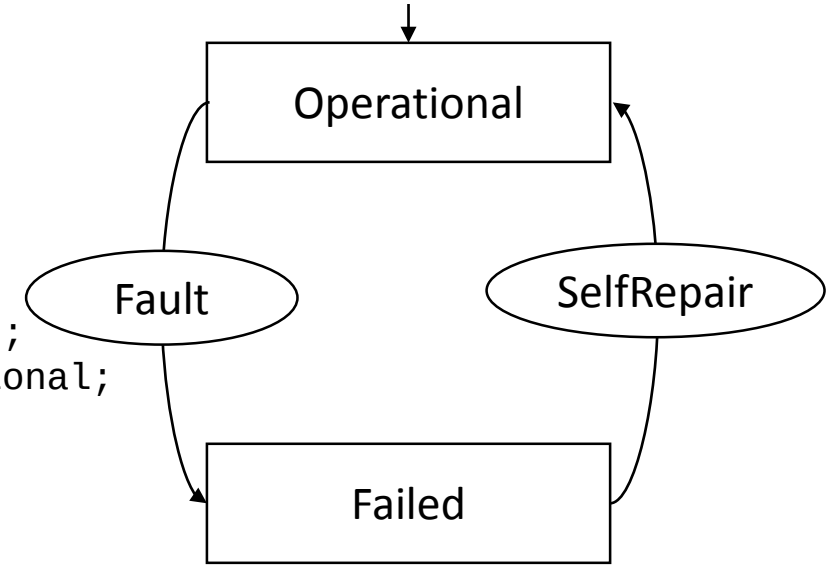
Can be reused in any component

OSATE includes a set of reusable state machines

See `ErrorLibrary.aadl`




```
annex EMV2 {**
error behavior ExampleBehavior
    events
      Fault: error event;
      SelfRepair: recover event;
states
      Operational: initial state ;
      Failed: state;
transitions
      SelfFail: Operational -[Fault]-> Failed;
      Recover: Failed -[SelfRepair]-> Operational;
end behavior;
**};
```

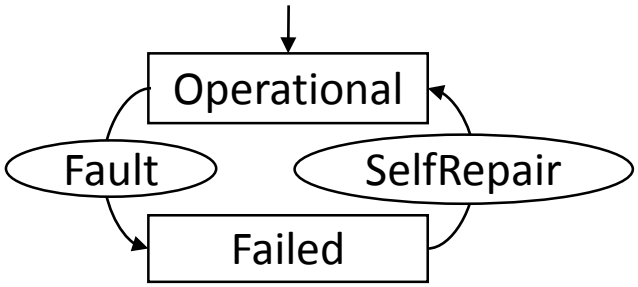


Component Error Behavior Specification

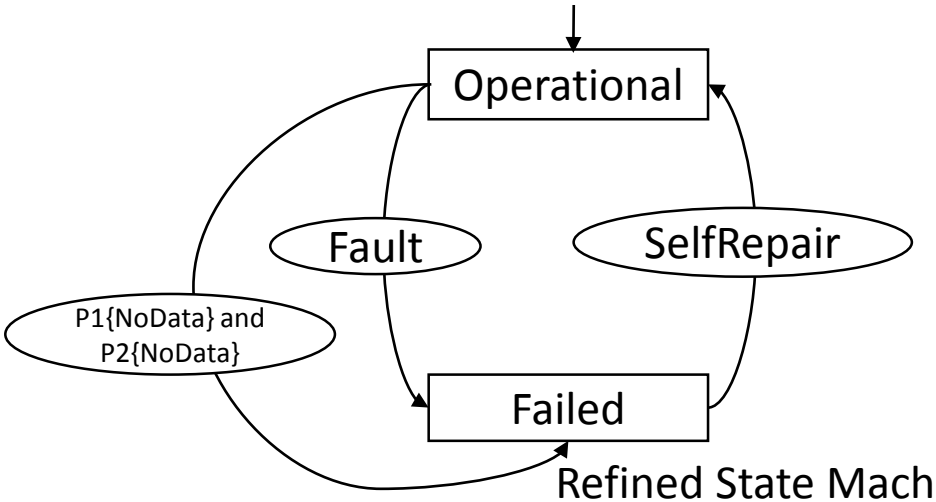
Component-specific behavior specification

- Identifies an generic error behavior state machine and uses it on this component
- Optionally refine the state machine with component-specific declarations: additional events, transitions, etc.

Used in Fault-Tree Analysis



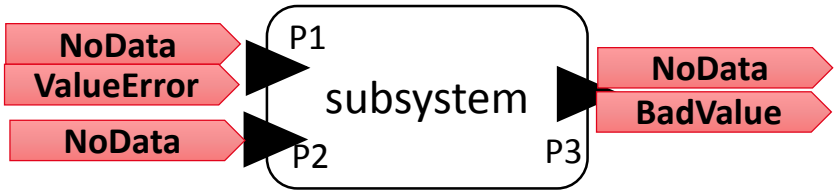
Generic State Machine



Refined State Machine

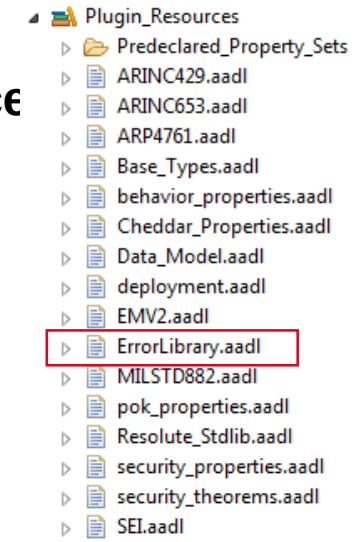
```
system mysystem
annex EMV2 {**
   use types ErrorLibrary ;
   use behavior MyErrorLibrary::ExampleBehavior ;

   component error behavior
   transitions  -- additional transitions that are component specific
       Operational-[P1{NoData} and Port2{NoError}]->Failed;
end behavior;
```



Studying existing pre-declared State Machines

Open OSATE and the ErrorLibrary.aadl file in the Plugin_Resource project. Please answer to the following answers (5 minutes)



1. What is the difference between FailStop and DegradedFailStop?

2. In DegradedRecovery, what are the names of the states?
   How many failures are required to be in an unrecoverable state?

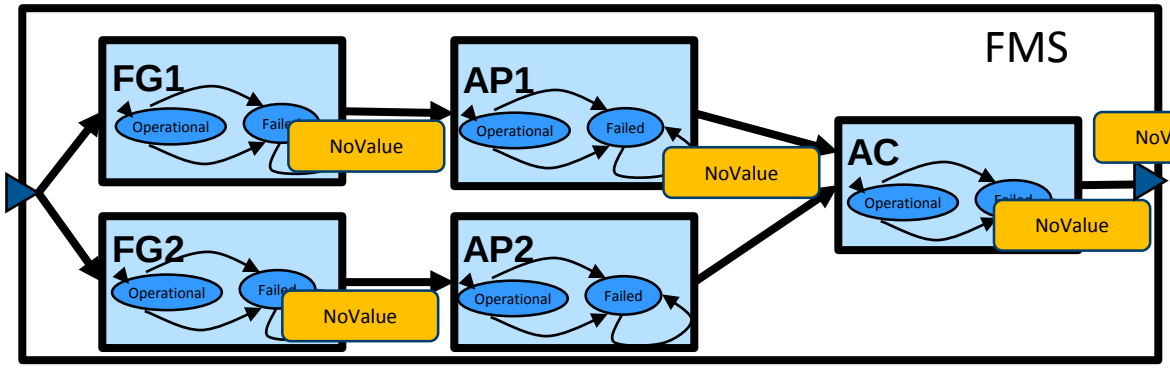
3. In PermanentTransientFailure, what is the condition to Fail and not recover?
   What is the probability to end up in that state?

Composite Error Model

State component = f (state subcomponent(s))

Can be used only in component implementation (needs subcomponent)

Used in the Fault-Tree Analysis




```
system implementation fms.i
subcomponents
   fg1 : system fg;
   fg2 : system fg;
   ap1 : system ap;
   ap2 : system ap;
   ac  : system ac;
annex EMV2 {**
   use types errorlibrary;
   use behavior errorlibrary::failstop;

   composite error behavior
   states
      [fg1.failed and fg2.failed]-> failed;
      [ap1.failed and ap2.failed]-> failed;
   end composite;
**};
end fms.i;
```

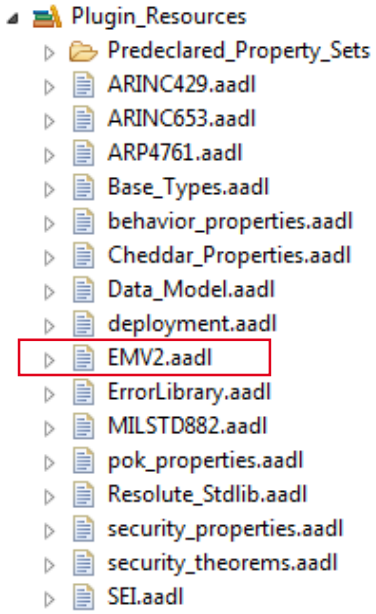
The fms system will fail if

1. **fg1** and **fg2** are in the state failed

OR

2. **ap1** and **ap2** are in the state failed

Safety-specific properties



Bring properties mechanisms to safety-related modeling elements

Extend modeling notation

Standard OSATE distribution includes several properties - see EMV2.aadl

Important properties, used by analysis tools

- **`EMV2::OccurrenceDistribution`** – probability of a failure/fault used by FHA & FTA
- **`EMV2::Severity`** and **`EMV2::Likelihood`** severity/likelihood of the failure, refined for specific standards - used by FHA
- **`EMV2::Hazards`** – describe/comment the failure - used by FHA

Property usage

```
device sensor annex EMV2 {**
   use types errorlibrary;
   use behavior errorlibrary::failstop;

   error propagations
      temp : out propagation {latedelivery,outofbounds};
   flows
      ef0 : error source temp{latedelivery};
      ef1 : error source temp{outofbounds};
   end propagations;

   properties
      emv2::severity   => ARP4761::Major applies to ef0,ef1;
      emv2::likelihood => ARP4761::Probable applies to ef0,ef1;

      emv2::hazards    =>
        ([crossreference => "N/A";
         failure => "Late Value";
         phases => ("all");
         description => "Late delivery. The value still arrives.";
         comment => "";
         ]) applies to ef0;

      emv2::OccurrenceDistribution =>
        [ProbabilityValue => 0.1 ;
         Distribution => Fixed;] applies to ef0;
**};
end sensor;
```

Automation of SAE ARP4761 System Safety Assessment Practice

AADL

one consistent model
multiple analysis

FHA

Spreadsheet

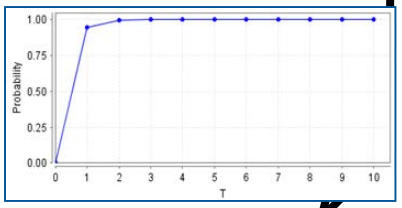
Uses error sources

Component	Error	Hazard Description	Crossrefer	Functional Failure	Operational P
StabilatorPositionSe	"ServiceOmission o	"No stabilator position readings due to s	"1.1.3"	"Loss of sensor readings"	"all"
StabAct1	"ServiceOmission o	"Failure to move stabilator into desired	"1.1.2"	"Loss of actuator functionalit	"all"
StabAct2	"ServiceOmission o	"Failure to move stabilator into desired	"1.1.2"	"Loss of actuator functionalit	"all"
StabilatorController	"null on ActCmd"	"Absence of computed data should signa	"1.1.1"	"Loss of guidance values"	"Approach"
StabilatorController	"null on ActCmd"	"Absence of computed data should signa	"1.1.1"	"Loss of guidance values"	"Approach"
StabilatorController	"null on ActCmd"	"Absence of computed data should signa	"1.1.1"	"Loss of guidance values"	"Approach"

Markov Chain

PRISM

Uses error flows & behavior



FMEA

Spreadsheet

Uses error flows & propagations

Three CPU FMEA

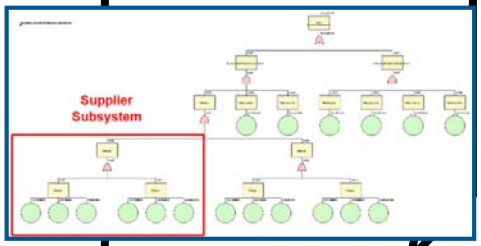
Date: 12/1/2010 Author:

Item	Initial State	Initial Failure	1st Level Effect	Transition	2nd Level Effect
CPU_1.cpu	ErrorFree	CPU_Failure	PermanentError	out_CPU_Failed	PermanentError
PR_AP_L	ErrorFree		ErrorFree	CPU_Failed(N)	PermanentError
CPU_2.cpu	ErrorFree		ErrorFree		ErrorFree
PR_AP_R	ErrorFree		ErrorFree		ErrorFree
PR_FGS_L1	ErrorFree		ErrorFree		ErrorFree
CPU_3.cpu	ErrorFree		ErrorFree		ErrorFree
PR_FGS_R1	ErrorFree		ErrorFree		ErrorFree
CPU_1.cpu	ErrorFree		ErrorFree		ErrorFree
PR_AP_L	ErrorFree		ErrorFree		ErrorFree
CPU_2.cpu	ErrorFree	CPU_Failure	PermanentError	out_CPU_Failed	PermanentError
PR_AP_R	ErrorFree		ErrorFree	CPU_Failed(N)	PermanentError
PR_FGS_L1	ErrorFree		ErrorFree	CPU_Failed(N)	PermanentError
CPU_3.cpu	ErrorFree		ErrorFree		ErrorFree
PR_FGS_R1	ErrorFree		ErrorFree		ErrorFree

FTA

CAFTA, OpenFTA

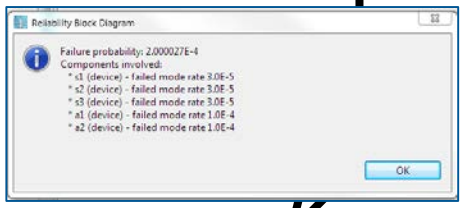
Uses composite error behavior



RBD/DD

OSATE plugin

Uses composite error behavior



Exercise 4 - Objectives

Specify error flow source and sink for devices (`devices.aadl`)

- the `sensor` has two `error sources` through the `temp` features
  - `ef0`: error source for `latedelivery`
  - `ef1`: error source for `outofbounds`
- `cooler` and `heater` have two error sink in the `operate` feature
  - `ef0`: error sink for `latedelivery`
  - `ef1`: error sink for `serviceerror`

Specify comments on error sources for Functional Hazard Analysis (`devices.aadl`)

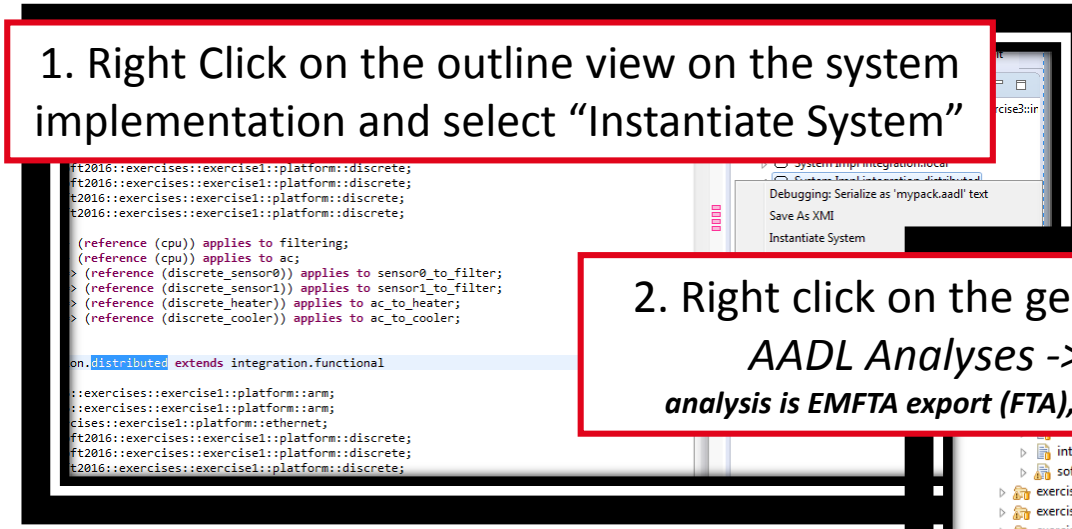
- Complete the `emv2::hazards` property on `ef0` and `ef1` for sensors

Generate and observe the analysis reports

- Generate Functional Hazard Assessment (FHA)
- Generate Fault-Impact (FMEA)
- Generate Fault-Tree Analysis (FTA)

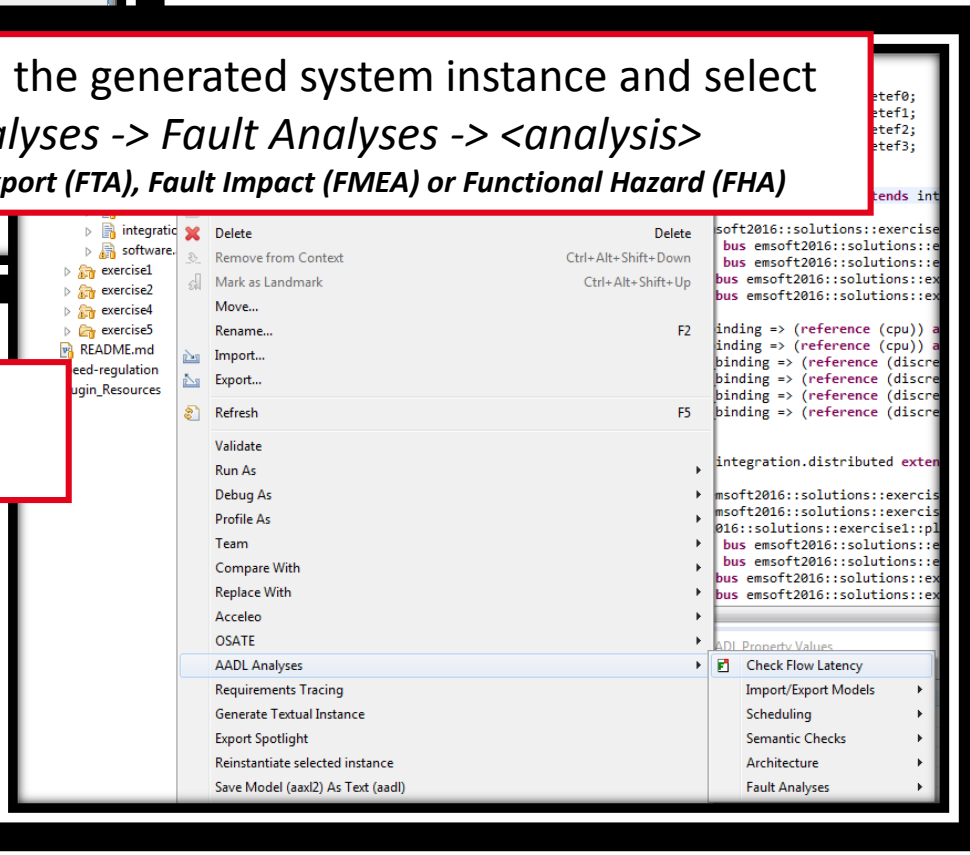
Exercise 4 – Generating Safety Reports

1. Right Click on the outline view on the system implementation and select “Instantiate System”



2. Right click on the generated system instance and select *AADL Analyses -> Fault Analyses -> <analysis>*

analysis is EMFTA export (FTA), Fault Impact (FMEA) or Functional Hazard (FHA)



3. Open the generated report in the generated **`reports/`** directory