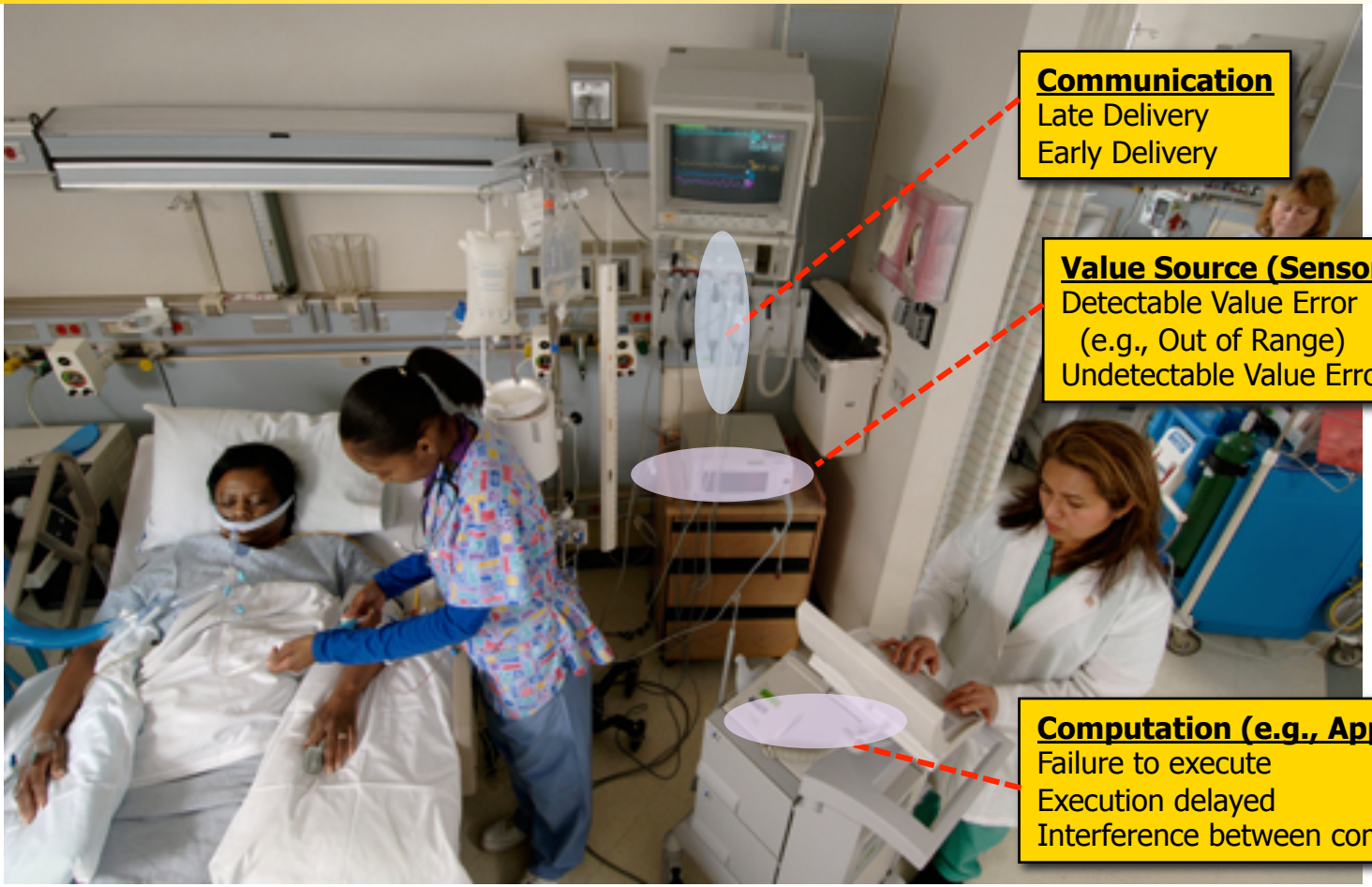
CIS 890: High-Assurance Systems

Hazard Analysis

Lecture: Error Modeling Annex
Version 2 – Error Types

Identifying Common Hazards

A common approach in safety standards is to identify different aspects/dimensions of a system and list common faults/hazards associated those aspects. These become "thought triggers" (see examples below) that lead one to assess if those hazards/faults could occur in their system and to the consider the potential harm of such hazards.



Review from Introduction

Error Modeling Annex - V2

- AADL annex to support hazard analysis
- And it offers
  - Error Type Hierarchy
  - Fault Propagation
  - Component Error Behavior
  - Composite Error Behavior

Fault Types / Common Hazards

As an example, IEC 80001 has a very "bare bones" list of issues. We anticipate that a significantly expanded list will be necessary to support our work.

1) Loss of function (compromised availability)
   a) Major loss of function
      i) Loss of data (loss of connectivity)
         1) Intermittent connectivity
         2) Complete loss of connectivity
      ii) Loss of function of MEDICAL DEVICE
         1) Incorrect data (compromised integrity)
         2) Incorrect data (PATIENT mismatch)
   b) Degraded function
      i) Incorrect or inappropriate timing of data
      ii) Incorrect or inappropriate data interchange or INTEROPERABILITY
      iii) Unintended interactions between endpoints
      iv) Degraded function of MEDICAL DEVICE
2) Loss of confidentiality
   a) Unauthorized access to data

IEC 80001 -- Appendix A -- *Common HAZARDS, HAZARDOUS SITUATIONS, and causes to consider in MEDICAL IT-NETWORKS*

EMV2 : Error Types

Overview

- Error Types and Type Sets
- Custom Error Types
- Common Fault Types : Ontology
  - Service-Related Error
  - Value-Related Error
  - Timing-Related Error
  - Replication-Related Error
  - Concurrency-Related Error
  - Authorization and Authentication-Related Error

EMV2 : Error Types

Introduction

- Error Type to characterize
  - Errors to be propagated
  - Activated fault represented by an error event
  - Failure mode represented by an error behavior

Error Type

```
error propagations
internal_failure: out propagation {ItemOmission, ItemCommission} ;
flows  --model error in detection and reporting
  inf: error source internal_failure{ItemOmission} when failed;
end propagations;
```

Error Type Set

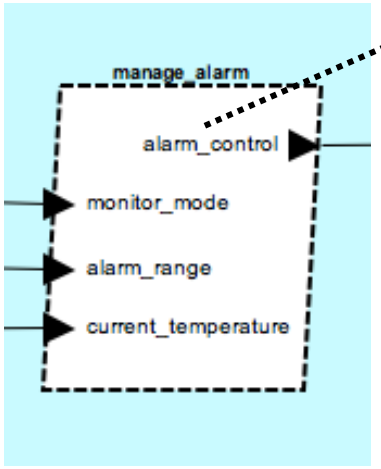
```
error types
   NoValue: type;
   BadValue: type;
   LateValue: type;
   MyTypes: type set { NoValue, BadValue, LateValue };
end types;

error behavior TypedStateMachine
use types TypedExampleLibrary;
events
   FailEvent: error event {MyTypes};
states
   Operational: initial state;
   FailedState: state {MyTypes};
transitions
   FailTransition: Operational -[FailEvent]-> FailedState;
end behavior;
**};
```

Error Type

EMV2 : Error Types

Error Propagation




```
error propagations
alarm_control: out propagation {AlarmError};
monitor_mode: in propagation {MonitorModeError};
--alarm_range is error-free
current_temperature: in propagation
  {OutOfRange,UndetectableValueError};
flows
  mame: error path monitor_mode{MonitorModeError}
    -> alarm_control{AlarmError};
  mact: error path current_temperature{UndetectableValueError}
    -> alarm_control{AlarmError};
end propagations;
```

What happened to the OutofRange error on port current_temperature?

Port alarm_control propagates only AlarmError

OSATE will report that OutOfRange error is not handled

Flows agrees with the error propagation, ie. MonitorModeError is the only incoming error type for port monitor_mode

Error path specifies how the incoming error is transformed to an outgoing error

AADL:EMV2

Custom Error Types

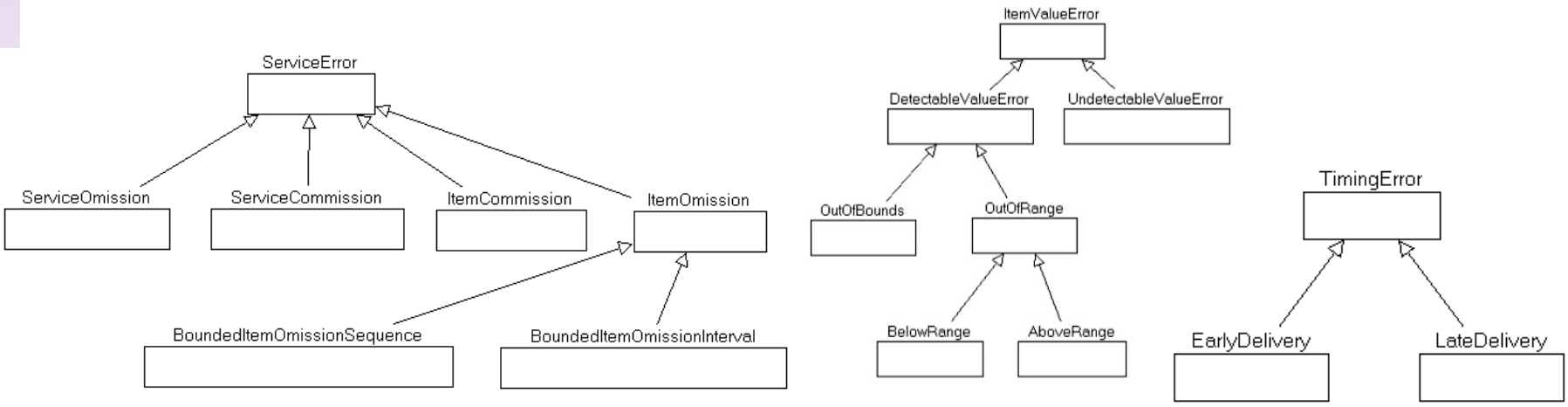
- Custom error types can be created in EMV2
  - To provide meaningful errors with respect to a component
  - Eg. HotAir vs AboveRange for HeatSource
- Three ways to create custom error types in EMV2
  - Completely new hierarchy
  - Extending an existing hierarchy
  - Renaming an existing error type

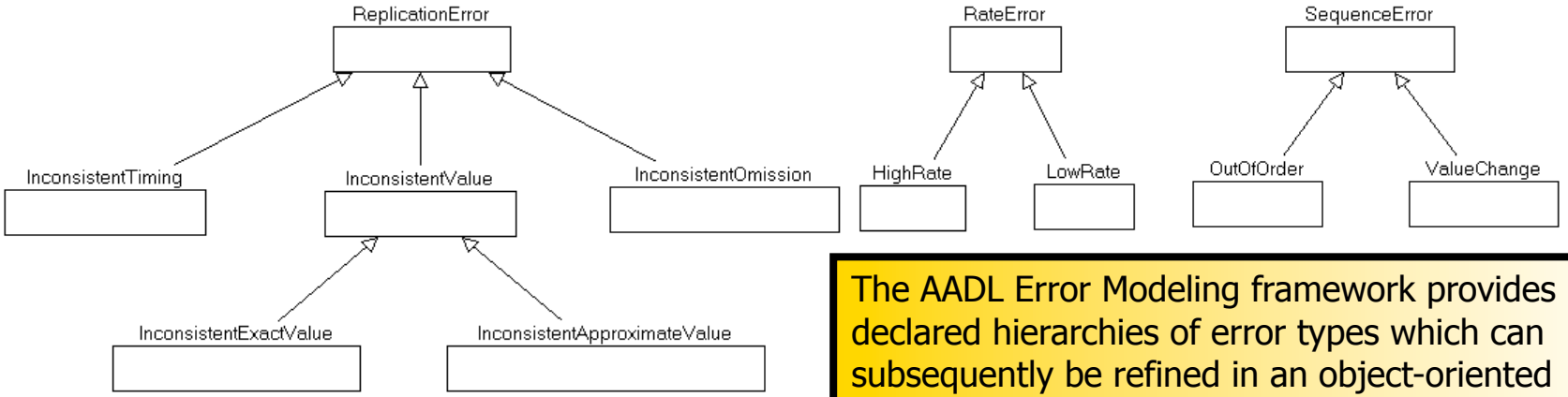
New error types by extending existing type

```
HeatControlError: type;  --heater on or off inappropriately
AlarmError : type;        --the class of alarm errors
FalseAlarm : type extends AlarmError;  --alarm erroneously sounded
MissedAlarm : type extends AlarmError;  --alarm missed
StatusError : type;  --mode and status errors
RegulatorStatusError : type extends StatusError;  --indicated regulator status wrong
RegulatorModeError : type extends StatusError;  --regulator mode wrong
MonitorStatusError : type extends StatusError;  --indicated monitor status wrong
MonitorModeError : type extends StatusError;  --monitor mode wrong
ThreadFault renames type ErrorLibrary::EarlyServiceTermination;  --thread fault halts thread
InternalError : type;  --an internal error was detected
DetectedFault : type;  --fault was detected (and announced)
UndetectedFault : type;  --fault occurred, but not detected
```

Fault Types / Common Hazards

One source of inspiration is the SAE Standard Error Model Annex for the Architecture Analysis Definition Language (AADL)



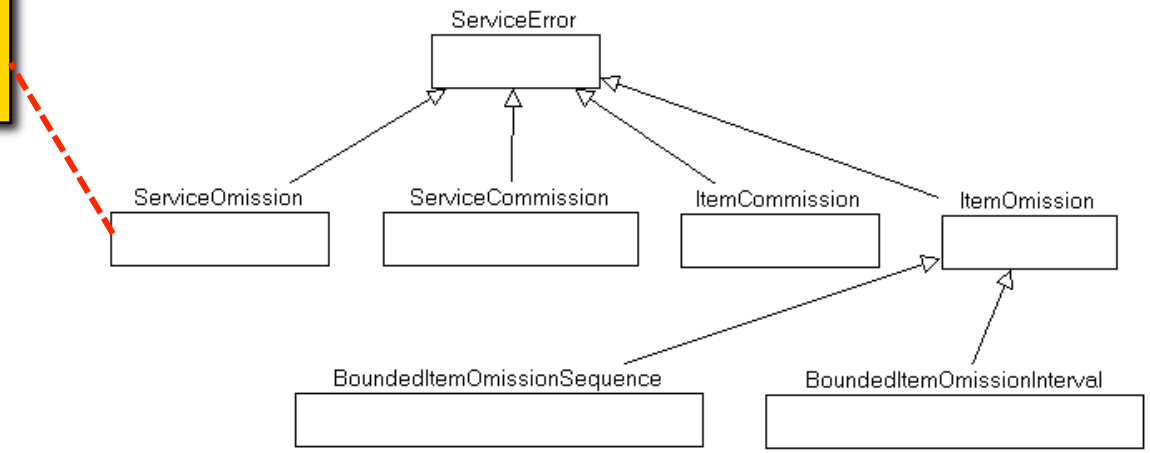


The AADL Error Modeling framework provides pre-declared hierarchies of error types which can subsequently be refined in an object-oriented fashion (e.g., *General* refined to *Particular*)

Fault Types / Common Hazards

The declarations on the previous page also have a textual representation…

Example of Service Omission
Temperature sensor fails to send sensed temperature



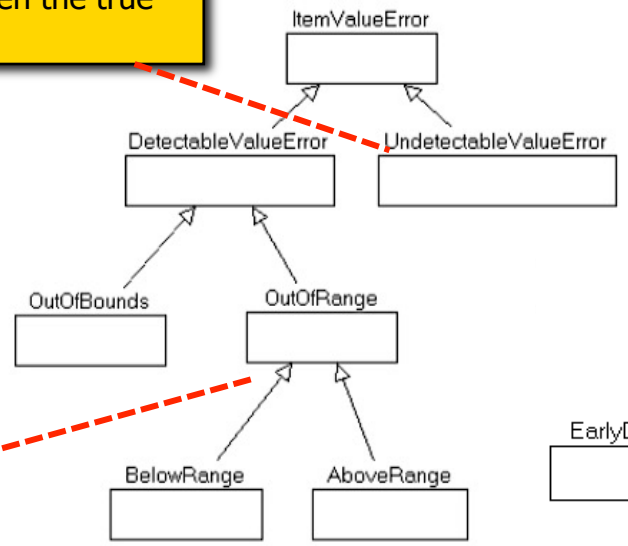

```
ServiceError:                type;
ServiceOmission:             type extends ServiceError;
ServiceCommission:           type extends ServiceError;
ItemComission:               type extends ServiceError;
ItemOmission:                type extends ServiceError;

BoundedItemOmissionSequence type extends ItemOmission;
BoundedItemOmissionInterval type extends ItemOmission;
```

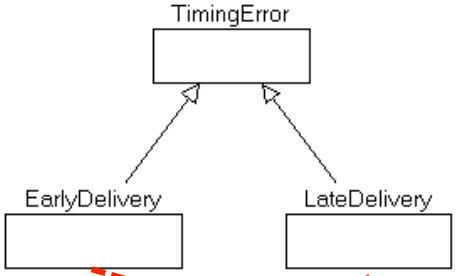
Fault Types / Common Hazards

Additional examples…

Example of Undetected Value Error
Temperature sensor sends a reading of 36.8 ℃ when the true reading is 38 ℃



Example of Detectable Value Error
Temperature sensor sends a reading of 110 ℃ (note: this value should never occur, and so this fault can be detected by a run-check)



Example of Timing
Thermostat responds late/early

Fault Types / Common Hazards

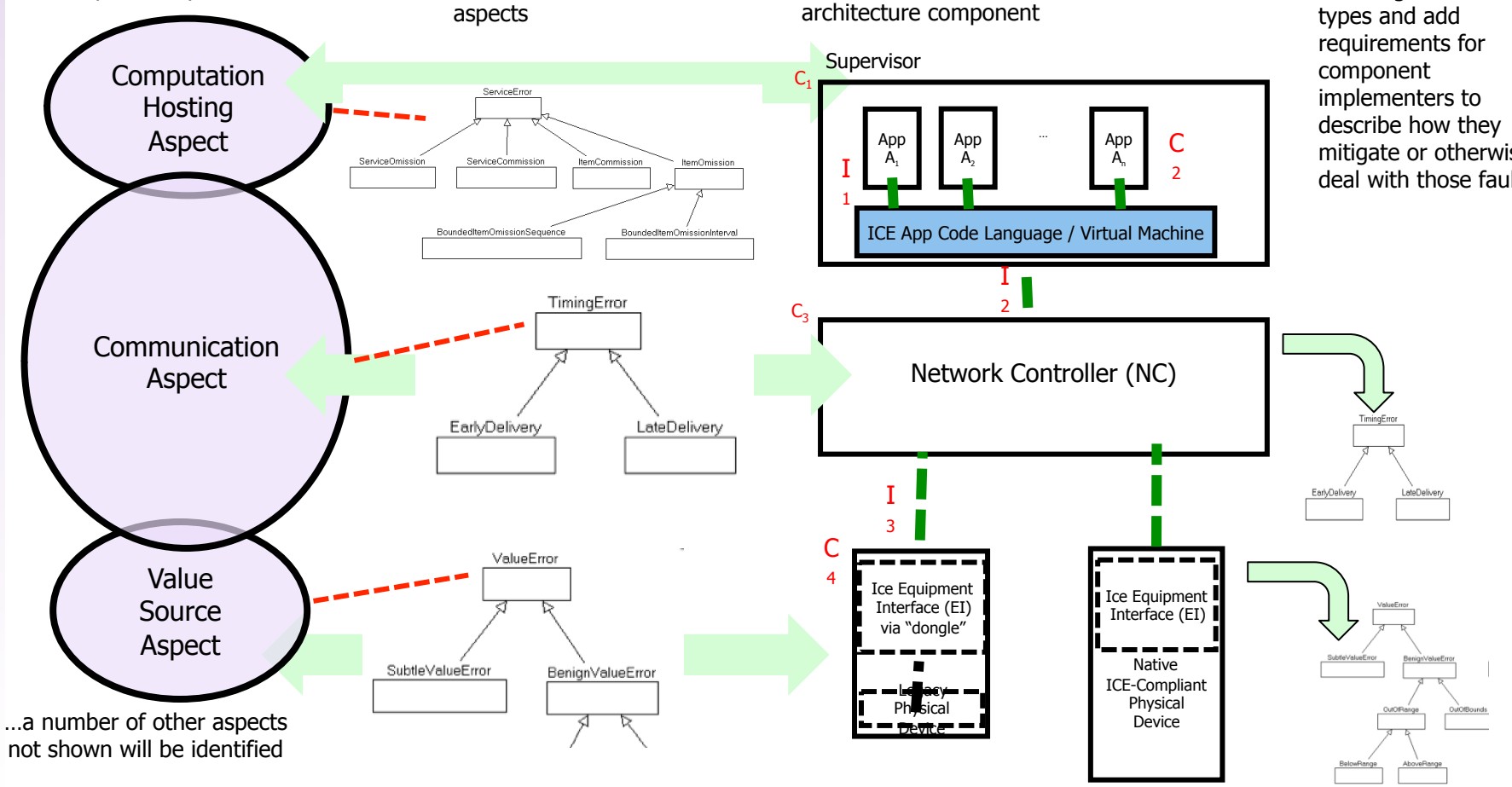
We are investigating an approach where *General Requirements* identifies different aspects/dimensions of a system and list common faults/hazards associated those aspects – subsequently refined to specific architecture and components in *Particular Requirements*

I. *General Requirements* will identity common functional aspects in an interoperable system

II. *General Requirements* will identity common fault types / hazards associated with those aspects

III. *Particular Requirements* will identify functional aspects (e.g, communication, value source) associated with each architecture component

IV. *Particular Requirements* refine/ extend general fault types and add requirements for component implementers to describe how they mitigate or otherwise deal with those faults



…a number of other aspects not shown will be identified

AADL Fault Type Definitions

Examples of AADL Fault Type Definitions…

(13) *Service errors* [`ServiceError`] are errors with respect to the number of service items delivered by a service. We distinguish between *Service Omission* errors to represent service items not delivered, and *Service Commission* errors to represent delivery of service items that were not expected to be delivered.

(14) *Service Omission* [`ServiceOmission`] represents an error where no service items are delivered.

Service Omission: $\forall \; s_i \in S \mid s_i = \varepsilon$ where ε is the empty service item.

(15) *Item Omission* [`ItemOmission`] represents an error where a single service item is not delivered.

Item Omission: $\exists \; s_i \in S \mid s_i = \varepsilon$ where ε is the empty action.

(16) *Bounded Omission Sequence* [`BoundedOmissionSequence`] represents an error where a certain number of consecutive service item omissions occur. A parameter *k* specifies the number of consecutive item omissions. For example, the CRC on satellite transmission allows some lost packets, but beyond the limit of the CRC, further packet loss causes loss of communication.

Bounded Omission Sequence error: $\exists \; [s_i \; s_{i+k-1}] \subset S \mid \; \forall \; s_j \in [s_i \; s_{i+k-1}] \mid s_j = \varepsilon.$

AADL Fault Type Definitions

(24) Value related errors deal with the value domain of a service. We distinguish between value errors of individual service items [`ItemValueError`], value errors that relate to the sequence of service items [`SequenceValueError`], and value errors related to the service as a whole [`ServiceValueError`]. Each is the root of a separate type hierarchy allowing us to characterize them independently, e.g., to specify that we have a `BoundedValueChange` error that may be `OutOfRange`. Note that both sequence and service value errors imply item value errors. Therefore, the type `ItemValueError` represents individual service item value errors that are singletons, i.e., not already covered by `SequenceValueError` and `ServiceValueError`.

(25) *Item Value Error* [`ItemValueError`] represents any kind of erroneous value for an individual service item.

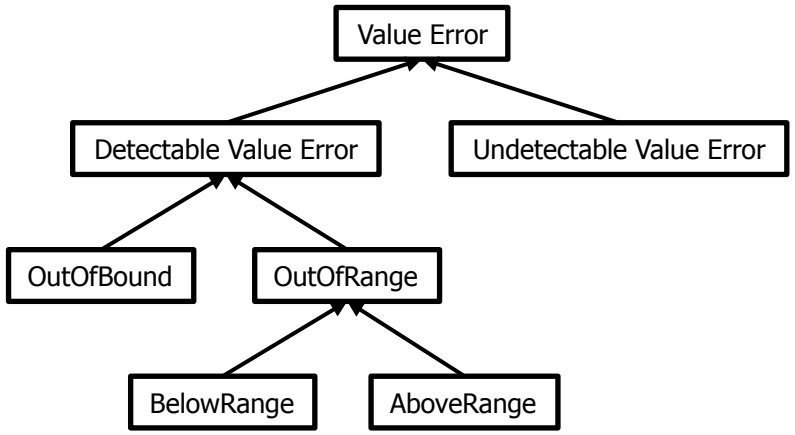
Item Value Error: $\exists\ s_i \in S \mid v_i \notin V_i$

(26) We distinguish between detectable and undetectable item value errors.

(27) *Detectable Value Error* [`DetectableValueError`] is detectable from the value itself, perhaps because it's out of range or has parity error. Let predicate B represent detection of a value error, B(v).

AADL:EMV2

Value Error Hierarchy



- Value Error: $\exists\, s_i \in S \mid v_i \neq V_i$
- OutOfRange: Values that are outside of a component's specification
- OutOfBound: The current item's value is above or below the acceptable range of values

Summary

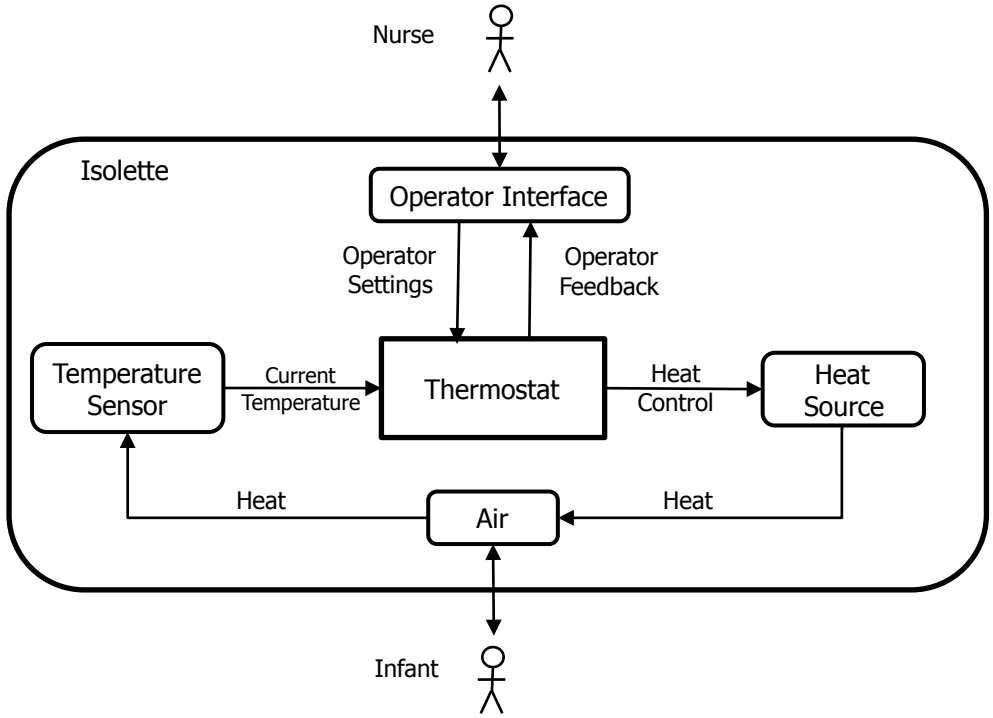
- Architecture-centric hazard analysis
- Incremental design and evaluation
- Error Types
- Fault Propagation
- Component and Composite behavior

AADL:EMV2

Language Concepts in EMV2

- Error Type system
  - Error types to characterize faults (source of error)
  - Predefined set of error types
  - Extendable and flexible to adapt
- Fault Propagation
  - Propagation of faults and their impact between components, systems and environment
- Component Behavior
  - Error states and transitions
  - Error trigger events
  - Error propagation and transformation
- Consistency between various HAT(FTA, FHA, FMEA)
- Alternate system architecture can be explored
- Changes are consistently propagated and reflected
- Hazard analysis can be incrementally performed

Isolette Model



AADL:EMV2 vs STPA

AADL:EMV2

- AADL Architecture
- Connections
- Error Types

STPA

- Control Structure
- Control Action
- Hazard Causes

EMV2 Definitions

Fault

Actual cause of a hazard

Error

Difference in state from the correct state

Failure

Deviation in behavior from a nominal specification

AADL:EMV2

Error Type System

EMV2 provides a flexible and hierarchical type system

Different hierarchical error types

- Value Errors
- Timing Errors
- Service Errors

AADL:EMV2

Error Type System

- Error is defined as the difference in state from the correct state.
- In EMV2 Error types are treated as a type set with a single element type

Service

- A service S is defined as a sequence of n service items si with n > 0
- A service item si is a pair (vi, di), where
  - vi is the value of the service and
  - di is the delivery time of the service

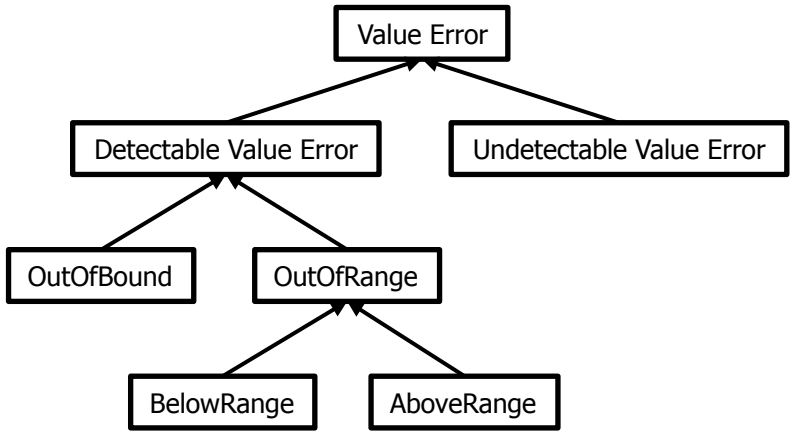
AADL:EMV2

No Error

- A service item(si) is defined as correct when
  - vi ∈ Vi and di ∈ Di, where
  - Vi is the correct range of values for vi, and
  - Di is the correct range of delivery time for di
- Challenge is knowing the correct values for a service item
  - For analysis: Assume there exist an oracle/application domain function providing correct values for si
  - Implementation: Redundant systems to identify correct values
- Note: There is an EMV2 keyword for NoError

AADL:EMV2

Value Error Hierarchy



- Value Error: $\exists\, s_i \in S \mid v_i \neq V_i$
- OutOfRange: Values that are outside of a component's specification
- OutOfBound: The current item's value is above or below the acceptable range of values

AADL:EMV2

Custom Error Types

- Custom error types can be created in EMV2
  - To provide meaningful errors with respect to a component
  - Eg. HotAir vs AboveRange for HeatSource
- Three ways to create custom error types in EMV2
  - Completely new hierarchy
  - Extending an existing hierarchy
  - Renaming an existing error type

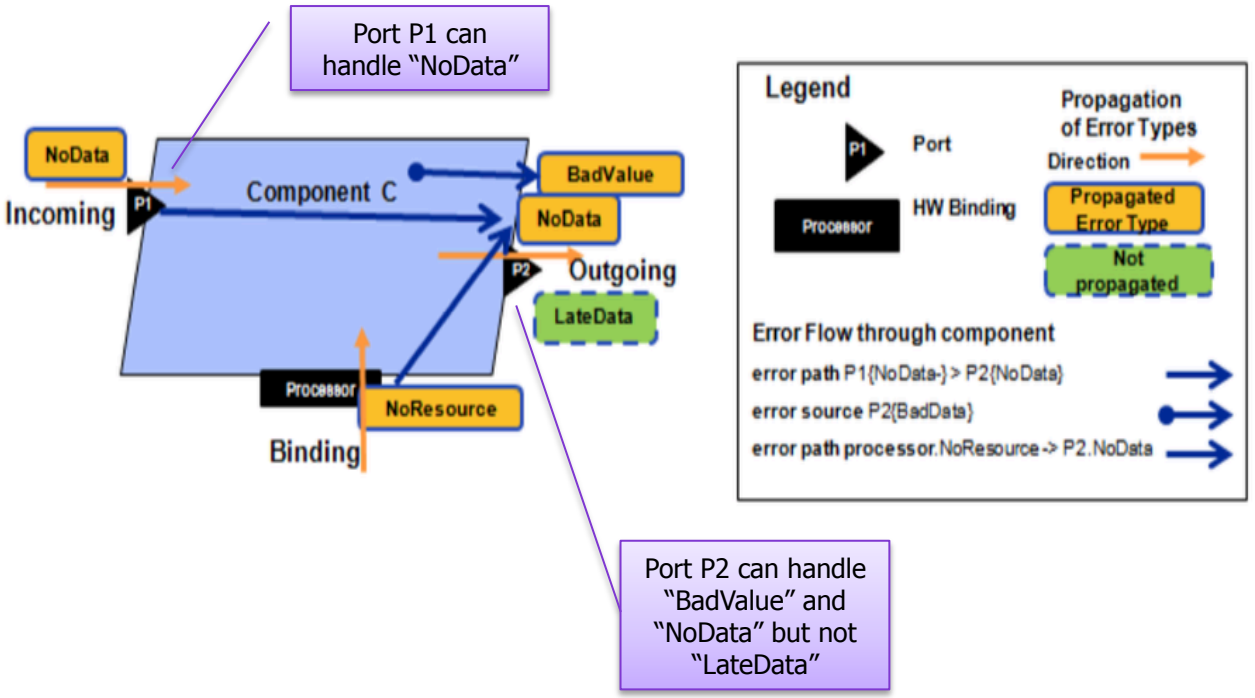
New error types by extending existing type

```
TimingError: type;
EarlyDelivery: type extends TimingError;
LateDelivery: type extends TimingError;
ValueError: type;
UndetectableValueError: type extends
  ValueError;
BenignValueError: type extends ValueError;
OutOfRange: type extends BenignValueError;
OutOfBounds: type extends BenignValueError;
BelowRange: type extends OutOfRange;
AboveRange: type extends OutOfRange;
```

AADL:EMV2

Component Error Propagation

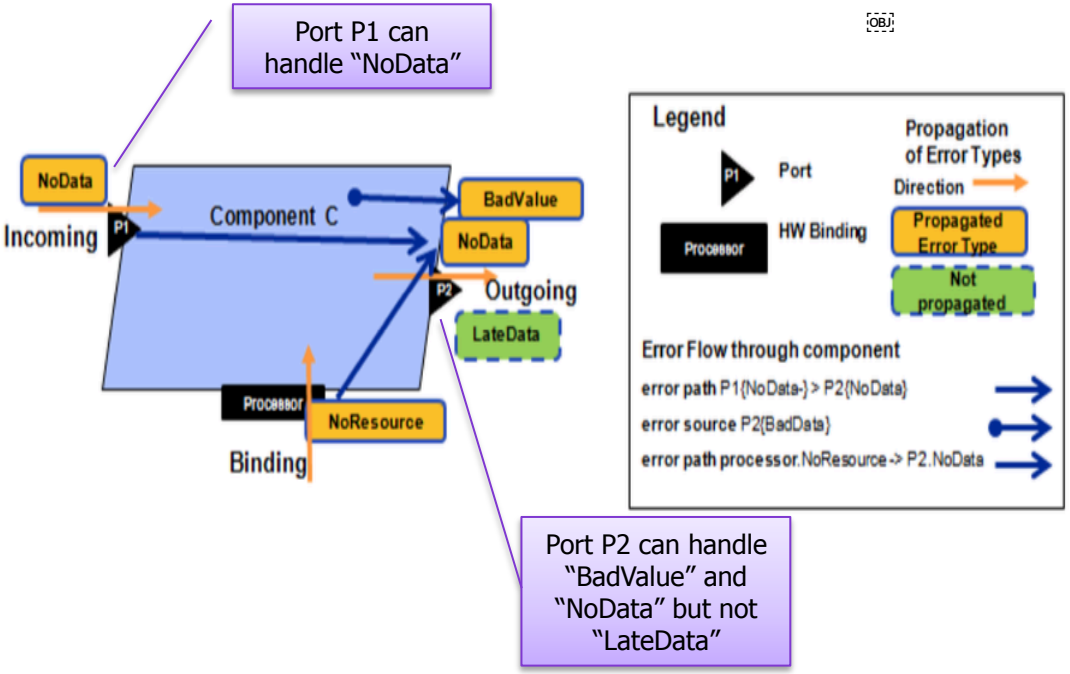
- Error propagation is like a error contract for the component
- It specifies the set of errors a particular port can handle



AADL:EMV2

Component Error Propagation

- Informs the set of possible incoming and outgoing errors
- It doesn't matter where the error originates
- Observing only at the boundary of a component
- In practice the there are derived from device specification

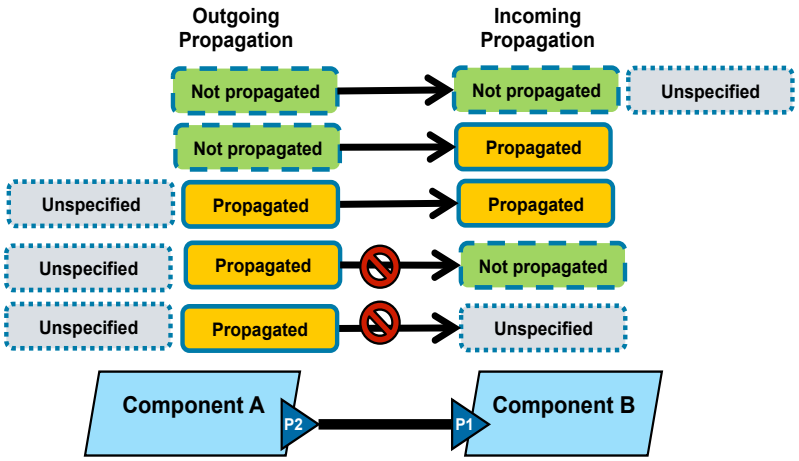


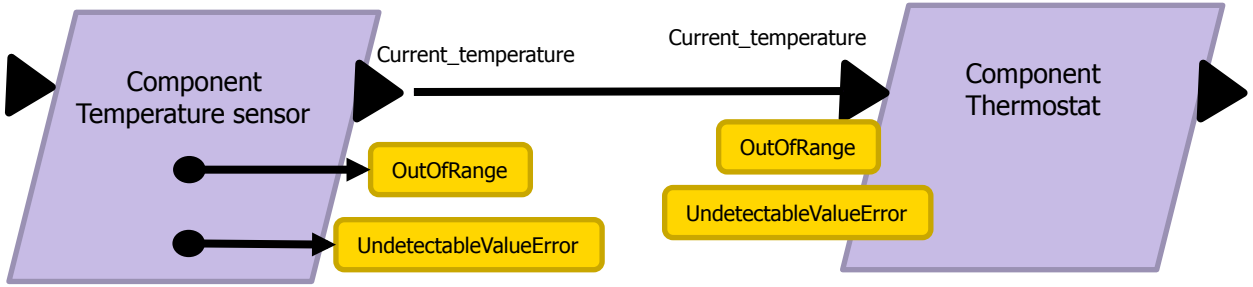

```
annex EMV2 {**
 use types ErrorLibrary;
error propagations
P1: in propagation {NoData} ;
P2: out propagation {NoData, BadValue};
P2: not out propagation {LateData};
end propagations; **};
```

AADL:EMV2

Consistency in Error Propagation

- To integrate two components A and B
  - If there exist a connection from port A.P1 to port B.P2, then
    $\forall e \in ErrorSet(A.P1) \mid e \in ErrorSet(B.P2)$
- Eg: Temperature sensor produces error "OutOfBound" and "UndetectableValueError"
- Also part of in propagation of Thermostat, therefore it is consistent

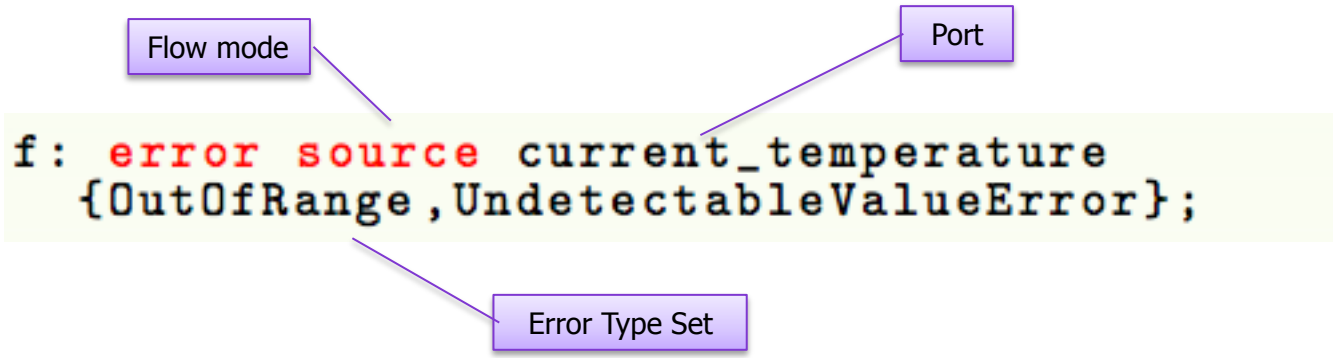




AADL:EMV2

Error Flow

- Specifies the relation between input and output ports
- It captures the error flow within a component
- Error flow can be
  - Source – Origin of an error
  - Sink – Containment / Suspension of error
  - Path – Transmission of error through component



AADL:EMV2

Error Flow : Path

- Describes the flow of error through a component
  - Propagation : Error A from in port is passed to out port
  - Transformation : Error A from in port is modified to error B and passed to out port

```
mmmiff: error path interface_failure
  {ItemOmission,ItemComission}
      -> monitor_mode(MonitorModeError);
mmminf: error path internal_failure
  {ItemOmission,ItemComission}
      -> monitor_mode(MonitorModeError);
mmmct: error path current_temperature
  {UndetectableValueError}
      -> monitor_mode(MonitorModeError);
```

All three path represents transformation

AADL:EMV2

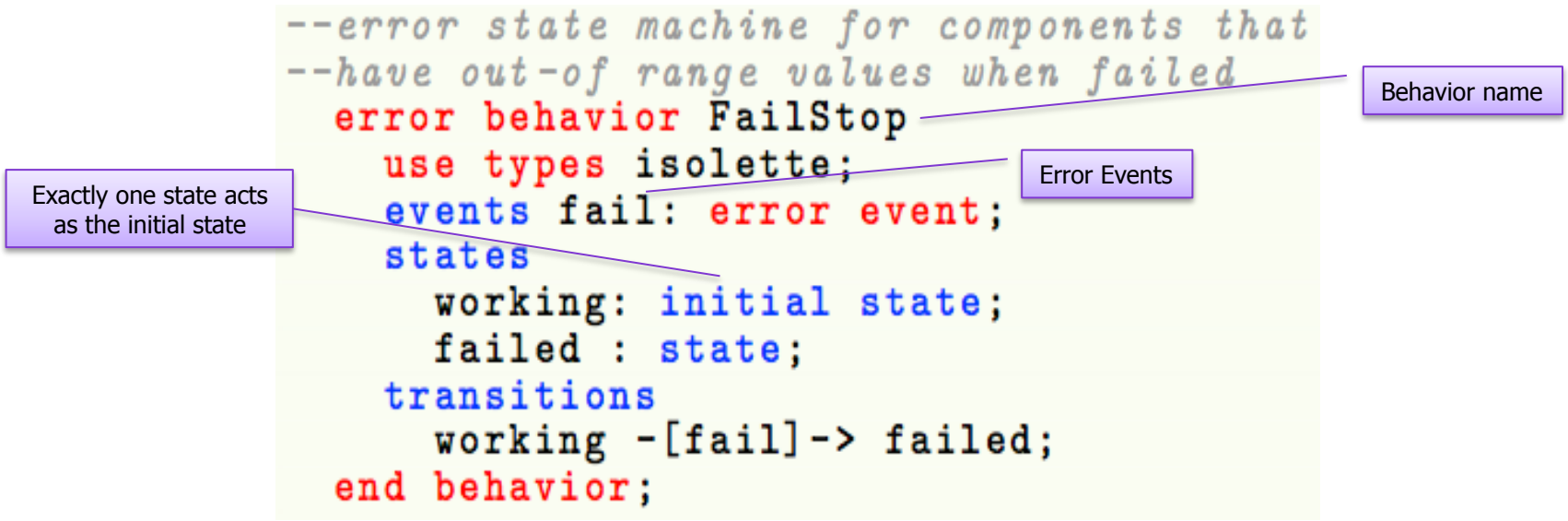
Component Error Behavior

- Error flow of dependents on the state of the component
- Error flow may be constrained based on the component state
- Three types of Behaviors:
  - Error behavior
  - Repair/mitigation behavior
  - Recovery behavior
- State transitions are triggered by error events and incoming error
- Similarly outgoing errors can be specified in terms of current state and incoming error
- Detected errors are mapped to a specific logging port with a error message/code

AADL:EMV2

Component Error Behavior

- Error behavior FailStop used in Isolette error detection mechanism



- FailStop behavior :
  - When the component is in "Working" state the "fail" even may trigger state transition to "failed" state

AADL:EMV2

Component Error Detection

- When an error is detected, the system may raise alarm or log error
- Component error behavior is used to specify detection condition

```
component error behavior
  detections
    failed -[ ]-> internal_failure!;
  --in "failed" state send event out
  --port internal_failure
end component;
```

The above specifies, when the component is in failed state, sends out an event through internal_failure port to raise alarm

AADL:EMV2

Composite Behavior

- Component's error behavior is specified in terms of its subcomponent

Error states of the component Isolette

```
error behavior CompositeFailure
  use types isolette;
  states
    Operational: initial state;
    ReportedFailure: state {DetectedFault};
    MissedFailure: state {MissedAlarm};
    FalseAlarm: state {FalseAlarm};
end behavior;
```

Error Types

Composite behavior: a constraint on the subcomponent's states to decide the component's state

```
composite error behavior
states
[temperature_sensor.failed
  or thermostat.ReportedFailure
  or heat_source.failed]->ReportedFailure;
[temperature_sensor.flakey
 or thermostat.MissedFailure]->MissedFailure;
end composite;
```