Outline

• Overview & Background
• Basic concepts illustrated
• Error Types
• Error Propagation
• Component Error Behavior
• Hierarchical composition of error behavior
• Error Behavior and Fault Management
• Dual Redundancy and Modes
• Error Behavior in Layered Architectures
• Summary
Terminology (Based on IFIP WG10.4)

Impairment is the inability to provide nominal behavior.

A fault is a root (phenomenological) cause of an error or a failure.

An error is the difference in state from a correct state.

A failure is a deviation in behavior from a nominal specification.

A propagation is the creation of a new error or failure due to an error or failure.

Observation

We use the term error in describing Error Model language constructs.
Background

Steve Vestal at Honeywell implemented an Error Model extension to MetaH in 1998.

SAE AS5506 Architecture Analysis and Design Language (AADL), a standard for describing embedded computer system (software and hardware) architectures, was issued November 2004 and revised in Nov 2009.

The AADL language and standard permit extensions and tailoring.

SAE AS 5506-1, AADL Annex Volume 1, was issued in 2006.
- Includes Error Model Annex as small variation of MetaH Error Model capability.

Revision of Error Model Annex in progress to improve its expressive power and semantics.
- Expected publication as part of AS-5506-3 Volume 3 in early 2012.
AADL Error Model Scope and Purpose

System safety process uses many individual methods and analyses, e.g.

- hazard analysis
- failure modes and effects analysis
- fault trees
- Markov processes

Related analyses are also useful for other purposes, e.g.

- maintainability
- availability
- integrity

Goal: a general facility for modeling fault/error/failure behaviors that can be used for several modeling and analysis activities.
Example: Integrated Safety, FMEA, Reliability

Capture hazards

System

Capture risk mitigation architecture

Subsystem

Capture FMEA model

Component

Error Model features permit checking for consistency and completeness between these various declarations.
Goal of the Error Model Annex Revision

A core set of reliability concepts and error types
Interaction of systems with nominal behavior and threats in the form of defects, faults, misbehavior, violated assumptions resulting in error propagations (hazards)
Anomalous and threat mitigation behavior of a system or a system component
Composability of error model in the system hierarchy

Error model representation improvements
• Add error semantics to stochastic communicating state machines
• Properties on all error model concepts
• Error types and attributes
• Clean separation of error propagation, error behavior, composition
• Explicit propagation and observation
Approach for Error Model Annex Revision

Utilize core AADL property sublanguage
Define hierarchy of error types (Ontology)
Focus on fault propagation
  • External fault interaction with other components
Focus on fault behavior inside component
  • How the component deals with its internal faults & incoming propagations
Focus on component hierarchy
  • System fault behavior in terms of subsystem fault behaviors
Focus on interaction between fault management specification & management implementation
  • Detection (heart beat, output monitoring)
  • Containment (space partition, time partition)
  • Restoration action: Recovery/repair behavior (replacement, extrapolation)
    – Isolation higher level symptom decomposed into lower level causes
Error Characteristics to be Reflected

Error event & recovery/repair event
Intrinsic error: defect, fault, misbehavior
  • Permanent, transient error (state)
  • Internal & propagated
  • Intentional & unintentional propagation
  • Unknown, Undocumented, Unhandled
  • Prevented errors (prevention hazards)
  • Mitigated errors (mitigation hazards)
  • Detected, reported, mitigated, propagated
  • Explicitly propagated & observable errors
Error Model and the Architecture

Propagation of errors of different types along propagation paths between architecture components.

Component error behavior as transitions between states triggered by error events and incoming propagations.

Error flows as abstractions of propagation through components.

Composite error behavior in terms of component error behavior states.
Error Model Annex Sublanguage

Error Model annex declarations

- **Reusable annex library declarations**
  - Error Types and Type Sets (V1 Error Model Types & Propagations)
  - Error Type Mappings (V1 In and Out Guards)
  - Error Behavior State Machines (V1 Error Model Implementations)

- **Component specific annex subclauses**
  - Error Propagations & Flows (V1 Propagations & In and Out Guards)
  - Component Error Behavior (V1 Abstract error model)
  - Composite Error Behavior (V1 Derived error states)
Outline

• Overview & Background
• Basic concepts illustrated
• Error Types
• Error Propagation
• Component Error Behavior
• Hierarchical composition of error behavior
• Error Behavior and Fault Management
• Dual Redundancy and Modes
• Error Behavior in Layered Architectures
• Summary
A Simple Error Model Illustrated

Collection of *error types*, used to specify *error propagations* and *flows* to support fault impact analysis such as Failure Mode and Effects Analysis (FMEA).

- Error propagations can represent hazards
- The propagation paths are determined by the architecture.
- Components can be the source or sink of error propagations, or pass errors on or transform them into different types of errors.
A Simple Error Model Illustrated

Components can have error 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*.

---

**Legend:**

- **Error propagation**
- **Error event**
- **Error flow**
- **Error propagation path**

© 2011 Carnegie Mellon University
Defining Error Types and Propagations

Error types are defined in the Error Model library

- **Error types**
  - NoValue: \texttt{type};
  - BadValue: \texttt{type};
  - EarlyValue: \texttt{type};
  - LateValue: \texttt{type};

Error propagations are defined for ports (features) and bindings of individual components

- Inport1: \texttt{in propagation} [NoValue, BadValue];
- OutPort2: \texttt{out propagation} [NoValue, BadValue, LateValue];

Error flows are specified between incoming and outgoing ports of components

- Flow1: \texttt{error source} Outport2 [LateValue];
- Flow2: \texttt{error path} Inport1 -> Outport2 [NoValue, BadValue];
Defining Error Behavior as State Machine

Error behavior state machines
• Defined in the Error Model library
• They define error/repair/recover events
  – BadValueEvent: error event;
  – NoValueEvent: error event;
  – LateValueEvent: error event;
• They define states and transitions
  – ErrorFree: initial state;
  – BadValueState: state;
  – NoValueState: state;
  – LateValueState: state;
  – BadValueTransition: ErrorFree-[BadValueEvent]-> BadValueState;
  – NoValueTransition: ErrorFree-[NoValueEvent]-> NoValueState;
  – LateTransition: ErrorFree-[LateValueEvent]->LateValueState;

The error states are mutually exclusive. The occurrence of BadValue and LateValue has to be modeled by a separate state. Transitions occur one error event at a time.
Defining Component Error Behavior

Defined by a Component Behavior subclause on component types and implementations

- Specify conditions for outgoing propagations in terms of current state and incoming propagations
  - **Propagation conditions**
    - Outport2[BadValue]: when in state BadValueState;
    - Outport2[NoValue]: when in state NoValueState;
    - Outport2[LateValue]: when in state LateValueState;
  - Specify conditions for transitions in terms of incoming propagations & events
    - **Transition conditions**
      - BadValueTransition when inport1[BadData];
      - NoValueTransition when inport1[NoValue] or processor[NoService];
  - Specify error behavior conditions that are detected by the system/component and reported as event
    - **Detection conditions**
      - Event self.DetectedNoValue: when in state NoValueState;
Use of Error Types on Events and States

Think of state machine with tokens of different types

- Error types can be placed into a type hierarchy
- Error events generate events of different types
- Event type becomes type of state
- Propagation type (trigger condition type) becomes type of state

Use of independent error types can be listed as type sets, as was done for propagation specifications of the initial simple model.

Placing types into the same type hierarchy allows us to specify that combinations of error cannot occur.
Typed Error Behavior State Machine

Compact state machine representation

- Types can be placed into type hierarchy
- Avoids replication of events, states, propagations for each error type
- Properties can be specified for each error type of an event

```
MyEvent: error event [MyType];
ErrorFree: initial state;
Failed: state [MyType];
EventTransition: ErrorFree -[MyEvent]-> Failed;

MyType: type;
NoValue: type extends MyType;
BadValue: type extends MyType;
LateValue: type extends MyType;

Transition conditions
EventTransition[BadValue]: when Inport1[BadValue];
EventTransition[NoValue]: when Inport1[NoValue]
or processor[NoService];

Propagation conditions
OutPort2: when in state Failed[MyType];
```

Defined in Error Model library

© 2011 Carnegie Mellon University
Error Event Handling in Error States

Transition for all events in every state
  • Implicit loopback vs. explicit specification

Event & in propagation coverage
  • Default interpretation: stay in state if no outgoing transition triggering
  • Potential for unhandled error event types
Outline

• Overview & Background
• Basic concepts illustrated
• Error Types
• Error Propagation
• Component Error Behavior
• Hierarchical composition of error behavior
• Error Behavior and Fault Management
• Dual Redundancy and Modes
• Error Behavior in Layered Architectures
• Summary
Error Type Definitions & Type Hierarchy

Error types are defined as reusable elements in the Error Model library
- NoPower: type;

Error types can be defined as subtypes of an existing error type
- Single inheritance type hierarchy
- Subtypes are mutually exclusive alternatives of a given error type
- Subtype is acceptable match for super type

Example ServiceError
- Service error is either service omission or service commission

```
ServiceError
  |
  |
ServiceOmission    ServiceCommission
```

© 2011 Carnegie Mellon University
Hierarchies of Independent Error Types

Independent error type hierarchies

- Below are some canonical error type hierarchies

Error types from different hierarchies can occur simultaneously
Error Types & Error Type Sets

Error type declarations

ServiceError: type ;
Omission: type extends ServiceError;
Commission: type extends ServiceError;
Early: type extends TimingError ;
Late: type extends TimingError ;

An error type set represents the combination of error types that can occur or be propagated simultaneously.

- An error type set is defined as the product of error types.
- Example: an error propagation may involve both a late value and an incorrect value.

InputOutputError : type set [TimingError, ValueError];
StreamError : type set [TimingError, ValueError, SequenceError, RateError];

An type set tuple represents a typed token instance

- Represents actual event, propagation, or state types
  [LateValue, BadValue] or [LateValue, noerror]
Formalized Error Type Specification

Service as sequence of service items

- Service S delivered by a system with a single user can be defined in terms of a sequence of service items, $s_i$, $i = 1, 2, ...$ each characterized by a tuple $v_{s_i}, t_{s_i}$ where $v_{s_i}$ is the value or content of service item $s_i$ and $t_{s_i}$ is the time interval or instant of observation of service item $s_i$.

- A service item is defined to be correct, i.e., have no error, iff: $(v_{s_i} \in SV_i) \land (t_{s_i} \in ST_i)$ where $SV_i$ and $ST_i$ are respectively the correct sets of values and times for service item $s_i$. $SV$ represents expected range of values throughout the service and $ST$ represent the expected duration of the service.

- Definition of service errors in the value and time domains as perceived by an omniscient observer of that service.
Service Related Error Specification

Service errors with respect to the service as a whole rather than individual service items

- **Service Omission** is perceived as a permanent fault in that no service items are provided after the point of failure.
  
  Service Omission: $\forall i, (\forall j \geq i, ts_j = \infty)$

- **Service Commission** is perceived as an impromptu service in that service items are provided before the point service is expected.
  
  Service Commission: $\forall i, (\forall j \geq i, ts_j \in ST_j)$

- Other forms of service error to be considered are: early service start, late service start, service duration too short, service duration too long.
Fault Mitigation Behavior

Based on these categories of error types we can specify desirable and undesirable fault and fault mitigation behavior. For example, we can define semi-consistent replicate value error behavior as Inconsistent Replicate Value error, where the value error of the replicate is a Corrupt Value error. The detection of the corrupt value by the recipient of the replicates ensures that all the non-corrupt values are identical with respect to the deployed error detection code.

• We can specify bounded omission behavior in that a system may omit some service items (Item Omission error), but if more than k items are omitted then all further items are omitted (Service Omission error).

• Bounded Item Omission error: $\forall i, (ts_i \in ST_i) \lor \forall j \geq i, ts_j = \infty \lor ((ts_i = \infty) \land (\exists j \in [i + 1, i + k]), ts_j \in ST_j)$
Fail Silent Behavior

We can specify fail-silent behavior of a system and delivery mechanism by assuming that the delivery mechanism can only introduce semi-consistent replicate value errors and that the system producing the service items delivers correct values. This leads to the follow fail-silent behavior with respect to the values:

• $\text{Fail-Silent } V_{FS} = \forall i, (\forall u \in [1,n], (vs_i \in SV_i) \land ((vs_i(u) = vs_i) \lor \lor (vs_i(u) \not\in CV_i)))$

With respect to time a system service always produces service items on time or stops producing service items. Similarly, the delivery mechanism is assumed to deliver the service items with consistent fixed propagation delays or stops delivering them.

• $\text{Fail-Silent } T_{FS} = \forall i, ((\forall j \geq i, \forall u \in [1,n], ts_j(u) = \infty) \lor \lor (\forall u,v \in [1,n], (ts_i(u) \in ST_i(u))) \land ((ts_i(u) - ts_i(v)) \leq \Delta)))$
Error Model Libraries & Namespaces

Error Model libraries and AADL Packages

- An AADL package can contain one Error Model library declaration
- The Error Model library is identified and referenced by the package name

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

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

Extend the error type hierarchy

• By adding subtypes anywhere in the type hierarchy

Accomplished by extending an Error Model library

• The library inherits all error types and type sets from the library being extended

Domain-specific aliases for error types

• Alias and aliased error type are considered equivalent

```
package MyErrorTypes
public
annex Error_Model {**
error types extends ErrorTypes with
    MissingData renames type ItemOmission;
    ExtraData renames type ItemCommission;
    WrongValue: type extends IncorrectValue;
    EstimatedValue: type extends IncorrectValue;
end types;
**};
end MyErrorTypes;
```
Matching of Error Types

Matching of error types

- Subtype acceptable match for a given super type
- Alias is an acceptable match for type
- Need for equivalence of error types in independent type hierarchies?
  - **NoPower and NoService** are declared alias of **Omission**
    - Either can be a match for **Omission**
    - Can they be matches of each other? yes
  - **NoPower and NoService** are declared subtypes of **Omission**
    - Either can be a match for **Omission**
    - Can they be matches of each other? no
Outline

• Overview & Background
• Basic concepts illustrated
• Error Types
• Error Propagation
• Component Error Behavior
• Hierarchical composition of error behavior
• Error Behavior and Fault Management
• Dual Redundancy and Modes
• Error Behavior in Layered Architectures
• Summary
An Overview of the AADL Error Model Annex

Error Propagation

Error Flow:
Path P1.NoData->P3.NoData
Source P2.BadData;
Path processor.NoResource -> P2.NoData

“Not“ indicates that this error type is not intended to be propagated.
This allows us to determine whether propagation specification is complete.

Incoming/Expected
• Propagated errors
• Errors not propagated
• Identification of missing specification

Bound resources
• Propagated errors
• Errors not propagated
• Identification of missing specification
• Propagation to resource

Outgoing/Intention
• Propagated errors
• Errors not propagated
• Identification of missing specification

© 2011 Carnegie Mellon University 33
Error Propagation
Error propagation is component-specific

- through component features & bindings
- declared in Error Model subclauses attached to component types and implementations
- The propagations for P1 and P3 specify unnamed type sets

```plaintext
system Subsystem
features
  P1: in data port; P2: in data port; P3: out data port;
annex Error_Model {**
    error propagations
    use types ErrorTypes;
    P1: in propagation [NoData, ValueError] ;
    P2: in propagation [ NoData];
    P2: not in propagation  [BadValue];
    P3: out propagation [NoData, BadValue];
    P3: not out propagation [LateData];
processor: in propagation [NoResource];
end propagations; **};
```
Imported Error Model Libraries

Imported Error Model libraries make error types and type sets accessible

- The namespace of an error model library is made visible by a `use types`
  - Types and type sets can be referred to without package name qualification
- Error types and type sets from multiple libraries can be made visible
  - Conflicting imported references must be qualified
  - A locally defined alias can be defined for a conflicting reference
Fault Propagation & Transformation Calculus

Build on York U. work
Leverage architecture model
Overcome short comings to address practical problems

Fault mapping rules

<table>
<thead>
<tr>
<th>*</th>
<th>late</th>
<th>(source)</th>
</tr>
</thead>
<tbody>
<tr>
<td>early</td>
<td>*</td>
<td>(sink)</td>
</tr>
<tr>
<td>omission</td>
<td>omission</td>
<td>(propagate)</td>
</tr>
<tr>
<td>late</td>
<td>value</td>
<td>(transform)</td>
</tr>
</tbody>
</table>

Default mapping
• Any to all

Single in port to multiple out ports
• late → (value, *, late)

Multiple in ports
• (late, _) → (value, late) : wildcard
• (late, f) → (f, late) : variable

Overlapping rules
• (*, late) → (*, *)
• (*, f) → (*, f)

Conditional mappings (considered too complex)
• (f, g) → late, if f = late
• * , if f = value and g = value
• value, if f = g = value
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 types of any incoming feature flows to all outgoing features
- If flows are specified in the core model error flows occur along those flows

Flows

F1: flow path p1 -> p3;

annex Error_Model {{
  error propagations
  ...
  flows
  es1: error source P3[BadData] ;
  es2: error source P3[NoData] ;
  error sink P2[NoData];
  ep1: error path F1[NoData];
  ep2: error path F1 mapping ErrorModelLibrary::MyMapping;
  ep3: error path processor -> P3 mapping ErrorModelLibrary::MyMapping;
end propagations ; **};
Propagation Paths are Determined by The Architecture

a processor to every thread bound to that processor
a processor to every connection routed through that processor
a memory to every software component bound to that memory
a memory to every connection routed through that memory
a bus to every connection routed through that bus
a device to every connection routed through that device
a component to each of its required and provided subcomponents
a component to everything that requires or provides it
a component to every connection from any of its out features
a connection to every component having an in feature to which it connects
a subcomponent to every other subcomponent of the same process
a process to every other process that is bound to any common processor or memory, except for processes that are partitioned from each other on all common resources
a connection to every other connection that is routed through any common bus, processor or memory, except for connections that are partitioned from each other on all common resources
an event connection to every mode transition that is labeled with an in event port that is a destination of that connection
Error Propagation Paths

Unintended propagations
Propagations due to residual faults
Not propagation may be violated

Mismatched Propagations
BadData -> not BadData
Binding Related Propagation Paths

On the SW component side we identify the binding point by the type of platform component

- assumes that there is only one instance of each being bound to (or the propagation from each of the instances does not need to be distinguished)

On the platform components side we can use the keyword `bindings` to identify the set of application component bound to the platform component
Two-way Propagation with Resources

- Bi-directional error propagation between HW and SW
- No enforcement of WCET
- Propagation from Processor
  - NoResource (Omission)
  - DeadlineMiss (Late) resource
- Propagation to Processor
  - ETOverrun (WCET)
  - Dispatch rate overrun
Propagation for Hardware Resources

Bindings
Reference to bound components

Access Connections
Directional access (access rights)

Power example
Use of bus for physical resources

Access keyword for bus or data subcomponent as access connection end
Properties on Propagations and Flows

Stochastic propagation behavior

• Occurrence probability & distribution on outgoing propagations
  – On flow sources to represent probability of an error event of a given type within the component resulting in a propagation
    • Occurrence probability can be specified for each error type
  – On flow sinks to represent the probability of an incoming propagation being masked
    • The same incoming propagation can also participate in a flow path for the same type or different types
    • Propagations that are not masked follow any flow path from the same incoming feature; if none are specified we have any to all flow paths
  – On flow path to represent probability of an incoming propagation of a specific type (or all types) following a given path
    • Different error types may have different Occurrence values
Propagation Probability and Flows

We have several kinds of probabilities

- **Probability of event occurrence**
  - Applied to flow sources to indicate the rate it generates events
  - Applied to out propagations if no flows are specified to represent the rate of out propagation
    - This represents the aggregate of flow source and paths

- **Probability of error propagation**
  - Applied to flow sinks to indicate the chance an incoming propagation is masked
    - The remaining errors are propagated on all paths
  - Applied to flow path to indicate the probability the error is propagated on the particular path
    - This allows different paths to have different propagation probabilities
Properties on Propagations, Events, Components

Hazards

- Severity: criticality level
- Description
- Likelihood
- Risk

Fault/failure

- Occurrence probability, distribution
- Persistency
- Duration
- Failure condition
Outline

- Overview & Background
- Basic concepts illustrated
- Error Types
- Error Propagation
- Component Error Behavior
- Hierarchical composition of error behavior
- Error Behavior and Fault Management
- Dual Redundancy and Modes
- Error Behavior in Layered Architectures
- Summary
Error Behavior State Machines

Reusable error behavior state machine (EBSM)

• Working, non-working states
• Named transitions with trigger conditions
• Destination branches for transitions with branch probabilities
• Error, recovery and repair events
• EBSMs can be refined through extends
Simple Error Behavior State Machine

annex Error_Model {**
    error behavior Example
    events  -- both events will have mode-specific occurrence values for powered,unpowered
        SelfCheckedFault: error event;
        UncoveredFault: error event;
        SelfRepair: recover event;
        Fix: repair event;
    states
        ErrorFree: initial state ;
        FailStopped: state;
        FailTransient: state;
        FailUnknown: state;
    transitions
        SelfFail: ErrorFree -[SelfCheckedFault]-> {FailStopped with 0.7, FailTransient with 0.3};
        Recover: FailTransient –[SelfRepair]-> ErrorFree;
        UncoveredFail: ErrorFree -[UncoveredFault]-> FailUnknown;
end behavior;
annex Error_Model {**

error behavior Example
use types TypesLibrary;

events -- both events will have mode-specific occurrence values for powered,unpowered
  SelfCheckedFault: error event [(BITFault, NoServiceFault)];
  UncoveredFault: error event [UndetectedFault];
SelfRepair: recover event;
  Fix: repair event;

states
  ErrorFree: initial state ;
  FailStopped: state [BITFault, NoServiceFault];
  FailTransient: state [BITFault];
  FailUnknown: state [UndetectedFault] ;

transitions
  BITFail: ErrorFree –[SelfCheckedFault[BITFault]]-> {FailStopped with 0.7, FailTransient with 0.3};
  NoServiceFail: ErrorFree -[SelfCheckedFault[NoServiceFault]]-> FailStopped ;
  NoServiceStop: FailStopped -[SelfCheckedFault[NoServiceFault]]-> FailStopped [NoServiceFault];
  Recover: FailTransient –[SelfRepair]-> ErrorFree;
  UncoveredFail: ErrorFree -[UncoveredFault]-> FailUnknown;

end behavior;
Working and Non-working States

annex Error_Model {**

error behavior Example

events -- both events will have mode-specific occurrence values for powered,unpowered

SelfCheckedFault: error event [ BITFault, UncoveredFault];
SelfRepair: recover event;
Fix: repair event;

states
ErrorFree: initial state {StateKind => working;};
FailStopped: state {StateKind => nonworking;};
FailTransient: state {StateKind => nonworking;};
FailUnknown: state {StateKind => nonworking;};

transitions
Fail: ErrorFree -[SelfCheckedFault]-> {FailStopped with 0.7, FailTransient with 0.3};
Recover: FailTransient –[SelfRepair]-> ErrorFree;
UncoveredFail: ErrorFree -[UncoveredFault]-> FailUnknown;

end behavior;
Component Error Behavior Specification

Component-specific behavior specification

• Identifies an error behavior state machine
• Specifies transition trigger conditions in terms of incoming propagated errors or working condition of connected component
• Specifies propagation conditions for outgoing propagated errors in terms of states & incoming propagated errors
• Specifies detection events, i.e., conditions under which an event is raised in the component (core AADL model)

component error model
use types MyErrorLibrary;
use behavior MyErrorLibrary::SWStateBehavior;
transition conditions
  Fail[BadData] when port1[BadData] and Port2[noerror];
propagation conditions
  Outport3[BadData] when 2 ormore (Port1[BadData], Port2[BadData], Port3[BadData]);
detection events
  Self.Failed event when FailedState;
properties
  Occurrence => 0.0005 Poisson applies to BadDataFault;
end component;
Component Error Behavior Specification

Component-specific behavior specification

- Identifies an error behavior state machine
- Specifies transition trigger conditions in terms of incoming propagated errors or working condition of connected component
- Specifies propagation conditions for outgoing propagated errors in terms of states & incoming propagated errors
- Specifies detection events, i.e., conditions under which an event is raised in the component (core AADL model)

```
component error model
    use types <librarynamelist>; -- makes error types visible for referencing error propagations
    use behavior error_behavior_state_machine_reference;
    [ transition conditions { transition_condition }+ ]
    [ propagation conditions { error_propagation_condition }+ ]
    [ detection events { detection_event }+ ]
    [ properties { error_modelcontained_property_association }+ ]
end component;
```
Transition Trigger Conditions

In V1 this was one of the roles of Guard_In

- Specification of a condition under which the named transition will occur
- The transition may already have specified a disjunction of error/repair events as trigger condition
- The transition already indicates the source state under which the transition will occur
- The transition trigger condition specifies under which incoming error propagation conditions the transition will occur
- Operators for boolean condition are: and, or, ormore, orless
- mask is an explicit specification that the component error state is not affected by incoming error propagations
  - The incoming error propagation may still affect outgoing error propagations as specified in a propagation condition
Transition Trigger Conditions - 2

Error types in states and in conditions

• Single event or incoming propagation as condition
  – Pass through of same type (default)
  Transition_identifier when SelfCheckedFault;
  Transition_identifier when port1[ValueError];

• Multiple incoming propagations & state
  – Explicit specification of resulting error type being assigned to state
  Transition_identifier [NoData] when port1[NoData] and Port2[NoData] ;
  Transition_identifier [BadData] when port1[BadData] or Port2[BadData];
  mask in state FailTransient [BadData] when <condition on incoming error propagations >;

Conjunctions on incoming propagations
Inport1[LateValue] vs. Inport1[LateValue] and inport2[EarlyValue] vs. Inport1[LateValue] and inport2[noerror]

noerror means that no error propagation occurs at a given time on a feature
Conditional Error Transitions

Uniqueness of triggered transition for multiple outgoing transitions

- Require uniqueness vs. condition ordering
- Atomicity of error events and error propagations
  - Example: error event triggers transition to Failed, while incoming error propagation transitions to TransientFailed
  - Event triggered transition specified with the transition requires that transition conditions for the same transition do not include the event
  - Different events are considered to occur at distinct times or atomic processing order is non-deterministic

Uniqueness of trigger (event or transition condition)

- Multiple events and/or transition conditions triggering the same transition
- Type sets allow combination of types to be assigned to a state
- Conflicting types within one type hierarchy may be resolved by type mapping rules or the triggers are processed atomically with events specified with transition taking precedence
Conjunctions on incoming propagations with single type

- `Inport1[LateValue] vs. Inport1[LateValue]` and `inport2[EarlyValue] vs. Inport1[LateValue]` and `inport2[noerror]`

- `noerror` means that no error propagation occurs at a given time on a feature

Conditions on incoming propagation type set tuples

- `Inport1[LateValue] vs. Inport1[LateValue,BadValue]` vs. `Inport1[LateValue,noerror]`

- Tuple without an element type, e.g., `ValueError` means any subtype and `noerror` matches
Error Propagation Conditions

Specification of a condition under which the outgoing propagation will occur

- In V1 this was one of the roles of Guard_Out
- The condition can be based on the error behavior state and on incoming error propagations or absence of a propagation (noerror)
- The condition may be solely based on an error behavior state
- The condition may be solely based on incoming error propagations
- Operators for boolean condition are: and, or, ormore, orless
- Mask on individual features (guard_out mask applied to features)
Error Propagation Conditions - 2

Error types in states and in conditions

• Single event or incoming propagation as condition
  Port1 in state FailedState[ValueError]; -- type inferred from state type
  Port1[BadValue] when Inport[BadData]; -- type mapped into propagated type

• Multiple incoming propagations & state
  Port1[NoData] in state Failed[NoData] when port1[NoData] and Port2[NoData];
  Port1[BadData] in state ErrorFree when port1[BadData] or Port2[BadData];

• Complex conditions
  Outport1[BadData] when Port1[noerror] and Port2[BadData] or Port1[BadData] and Port2[noerror];
  Outport1[LateData] when Port1[noerror] and Port2[LateData] or Port1[LateData] and Port2[noerror];
  Voter[BadData] when 2 orless (Port2[noerror], Port1[noerror], Port3[noerror]);
  Outport3[BadData] when 2 ormore (Port1[BadData], Port2[BadData], Port3[BadData]);

• Example of a port specific masking condition
  Outport2[mask] in state ErrorFree when (Port1[BadData] or Port2[BadData]);
Completeness and Consistency Checking

Coverage of error types in error propagation specification
  • Utilizes error propagation and contained error specification
  • Robustness and unintended error propagations

Consistency between error propagation, contained error, and error flow specifications and the error propagation conditions
  • Interactions between source/sink, contained error declarations (not), noerror, and mask
Modeling of Repair Actions

Interaction with health/fault management architecture
- Specification of error detection in system model
- Identification of repair actions in error model

Represent different repair behavior
- Traceability to repair action in actual system architecture
- Failure of repair action
- Repair action with duration
- Repair agent as shared resource
- Repair parts as consumable resource
Error Detection Conditions

In V1 this was one of the roles of Guard_Event

- Specification of a condition under which a failure (error) is detected by the system and reported as event
- The condition may be a single error state or an incoming error propagation or a logical condition of combinations
  - The component detects that it is in an error state
  - The component detects when it receives error propagations
  - Should we also include outgoing error propagations to model that a component detects when it propagates an error

EventOutPort when Port1[NoData] and Port2[NoData];
Self.Failed when in state FailedState;

Reference to modes, mode transitions, activation/deactivation in transition trigger conditions
Outline

• Overview & Background
• Error Types and Error Propagation
• Component Error Behavior
• Hierarchical composition of error behavior
• Error Behavior and Fault Management
• Dual Redundancy and Modes
• Error Behavior in Layered Architectures
• Summary
Hierarchical Modeling

A subsystem of components may have an explicitly associated error model.

The user may declare whether a subsystem error model

1. has a state determined by a user-specified function of the error states of the components (e.g. to model internal redundancy)

2. is an abstract error model to be substituted for the composition of the component models (e.g. to improve tractability of analysis)

The annex supports abstraction and mixed fidelity modeling.
Composite Error Behavior Specification

Composite error behavior specification for a component

- error behavior model in terms of the subcomponent error models
- Specifies conditions under which a composite state is the current state expressed in terms of the (current) error behavior state of its subcomponents.
- The logical expression has the operators **or**, **and**, **ormore**, **orless**.
- Must be consistent with abstracted error behavior specification

```plaintext
composite error behavior
use behavior qualified_error_state_machine_reference
composite states
Working when 2 ormore
    (sub1[ErrorFree], sub2.ErrorFree, sub3.ErrorFree);
CriticalFailure when 1 ormore
    (sub1.Failed[Critical], sub2.Failed[Critical], sub3.Failed[Critical]);
end composite;
```

Refer to state vs. refer to working/no-working. Also refer to type in state.
Outline

• Overview & Background
• Error Types and Error Propagation
• Component Error Behavior
• Hierarchical composition of error behavior
• Error Behavior and Fault Management
• Dual Redundancy and Modes
• Error Behavior in Layered Architectures
• Summary
Component & Composite Model for Dual Redundant System

- Dual redundant flight guidance system
  - Redundant Flight Guidance (FG) and Auto Pilot (AP) channel
- Two operational modes
  - Critical: requires both channels active
  - Non-critical: one channel active, operator initiated active channel selection
- Composite error model for FGS
  - Basis for system level reliability analysis
Two State Component Error Model

- State machine represents component failure
- Failure event & Failed state
- Component-specific propagation conditions for FG and AP

Package ErrorModelLibrary
public
annex Error_Model {**
  error types
  NoValue :type;
  end types;
  error behavior Simple
  events
  Failure: error event;
States
  ErrorFree: initial state;
  Failed: state;
Transitions
  BadValueTransition : ErrorFree -[Failure]--> Failed;
  end behavior;
**};
End ErrorModelLibrary;

-- FG component
component error behavior
  use types ErrorModelLibrary;
  use behavior ErrorModelLibrary::Simple;
  propagation conditions
  Outport[NoValue] when in state Failed;
  end component;

-- AP component
component error behavior
  use types ErrorModelLibrary;
  use behavior ErrorModelLibrary::Simple;
  propagation conditions
  Outport[NoValue] when in state Failed or
    in state ErrorFree and Inport[NoValue];
  end component;
Mode-specific Error Model Logic

- AC error behavior is sensitive to operational modes
  - Fail-stop behavior by not producing output (Omission error) under failure conditions

```plaintext
-- AC component
annex Error_Model {**
  component error behavior
  use types ErrorModelLibrary;
  use behavior ErrorModelLibrary::Simple;
  propagation conditions
    OutPort[NoData] when in state Failed or FromAP1Port[NoValue] or FromAP2Port[NoValue];
  **) in modes (Critical);
annex Error_Model {**
  component error behavior
  use types ErrorModelLibrary;
  use behavior ErrorModelLibrary::Simple;
  propagation conditions OutPort[NoValue] when
    in state Failed or (FromAP1Port[NoValue] and FromAP2Port[NoValue]);
  **) in modes (NonCritical);
```
Composite Error Behavior as Three State Model

- FGS state machine reflects failure in critical mode and non-critical mode
  - States are defined in terms of subcomponent states

```c
-- Annex subclause for FGS system
annex Error_model {**
  composite error behavior
    use types ErrorModelLibrary;
    use behavior ErrorModelLibrary::ThreeState;
  composite states
    ErrorFree when AP1 in state ErrorFree and AP2 in state ErrorFree and
    FG1 in state ErrorFree and FG2 in state ErrorFree and AC in state ErrorFree;
    CriticalModeFailure when 1 ormore (AC in state Failed,
      FG1 in state Failed, AP1 in state Failed,
      FG2 in state Failed, AP2 in state Failed);
    NonCriticalModeFailure when AC in state Failed or
      1 ormore (AP1 in state Failed, FG1 in state Failed)
      and 1 ormore (AP1 in state Failed, FG2 in state Failed);
  end composite;
**};
```
Outline

• Overview & Background
• Error Types and Error Propagation
• Component Error Behavior
• Hierarchical composition of error behavior
• Error Behavior and Fault Management
• Dual Redundancy and Modes
• Error Behavior in Layered Architectures
• Summary
Error Propagation Behavior of Connections

Bus/protocol impact on transfer
Propagated errors due to Bus/protocol
Mitigation of application error propagations

Connections do not have intrinsic error behavior, but are affected by error behavior of underlying protocol/bus
Connections and Bindings

Component A

Component B

Processor

Binding

Connection-Binding-
To buses, processors, devices, virtual buses

Processor 1

Processor 2

Bindings

Bindings Bus
Connection Error Propagation Scenarios

Source error types and binding propagation error types do not overlap

- The result tuple contains both and the connection destination must be aware of the binding propagation error types
  - Binding error type NoService
- The binding error type is mapped into an error type known to the destination
  - [NoData] when [NoService]
  - Must take into account whether source already propagates BadData
    - NoData when [BadData] and [NoService]

Source and binding propagation error types overlap

- For each overlapping error type we need to decide what the resulting type is for various combinations of source and binding type
- If the source is NoError then the result is the binding type

The resulting type involves multiple types of a type set from either the source or binding

- We need to match the source and binding tuple and identify the resulting tuple element types
  - [BadData, LateValue] when [BadData, noerror] and [LateXfer]
Transformation in Connection Error Propagation

Binding specific error types are mapped into error types known to recipient component

- Mappings attached to connection
- Different connections may

The bound virtual bus transforms source error propagations

- Example: BadData becomes NoData (Omission due to packet drop)
- Some BadData gets transformed while others remain BadData

The bound virtual bus masks source error propagations
We want specify how the error types being propagated from Component A to Component B get augmented by error types propagated from the protocol/bus/network.

Associate a set of type transformations with a component via annex subclause

Connections use transformations Library::MyTransformationSet;
Use of Error Type Transformations

Acceptable mappings between error types

- A mapping is triggered by a typed contributor (binding propagation)
- Mappings are inherent in the error types
- Non-overlapping mappings
  - Ambiguous overlap would require ordering of rules

- Applied to connections
  - Source is the connection source
  - Contributor is binding propagation
  - Target is the connection destination

Type transformations can also be applied to transitions of a state to itself. The contributors are the events and transition conditions result types.

<table>
<thead>
<tr>
<th>Source</th>
<th>Contributor</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoValue</td>
<td>NoValue</td>
<td>NoValue</td>
</tr>
<tr>
<td>NoValue</td>
<td>NoService</td>
<td>NoValue</td>
</tr>
<tr>
<td>NoValue</td>
<td>NoValue</td>
<td>(BadValue, LateValue)</td>
</tr>
<tr>
<td>BadValue</td>
<td>(BadValue, LateValue)</td>
<td>BadValue</td>
</tr>
<tr>
<td>BadValue</td>
<td>LateValue</td>
<td>LateValue</td>
</tr>
<tr>
<td>LateValue</td>
<td>LateValue</td>
<td>LateValue</td>
</tr>
</tbody>
</table>

BadValue : type;
NoValue : type;
LateValue : type;
NoService : type;
Bus Abstraction and Implemented As

Modeling Error Behavior of Networks as abstraction and as composite of its implementation.