ARTIST2 Summer School 2008 in Europe
Autrans (near Grenoble), France
September 8-12, 2008

Automating Compositional Safety Analysis for IMA Systems

Lecturer: Steve Vestal
Honeywell Labs

Outline

➡ System Safety

AADL Architecture and Error Modeling

Fault Trees

Generating Fault Trees from AADL Specifications

Referrals

System Safety Scope

- System safety is a discipline that applies across a range of products and processes.   Embedded computing is just one element within a broad system safety program.
- Risk identification, assessment and mitigation employ technical methods from many fields: fault-tolerance, verification, security, human factors, etc.
- Risk mitigation is not just reactive (e.g. fault-tolerance), it is also proactive (e.g. define safe operating procedures, provide functions that make overall operations safer).
- Safety is a process that begins before the product is built and continues throughout the lifetime of the product.

Basic Terms

- An **accident** or **mishap** is an undesirable event, e.g. that results in death, injury, damage to equipment or environment, or economic loss. Accidents are unplanned but not unexpected.
- A **cause** for an accident is a set of conditions that are individually necessary and collectively sufficient. An accident is the final event in a series of events, where interdiction at any point could have prevented that accident.
- A **hazard** is a state or set of conditions for a system that, together with unfavorable conditions in the environment, will lead inevitably to an accident. A hazardous state makes it impossible to prevent an accident.
- **Severity** is a measure of how bad an accident is, or how bad the accidents that might result from a hazard could be.
- **Likelihood** can be quantified as the probability that an accident of a given class and severity will occur (but the term may also be used in a qualitative or relative way).
- **Risk** is the combination of severity and likelihood.

MIL-STD-882D Standard Practice for System Safety

1. Document the system safety approach
2. Identify hazards
3. Assess mishap risk (combination of severity and likelihood)
4. Identify risk mitigations
   - Eliminate hazards
   - Incorporate safety devices
   - Incorporate warning devices
   - Define procedures and training
5. Reduce mishap risk to acceptable level (iterate)
6. Verify mishap risk reduction
7. Acceptance of residual risk by appropriate authority
8. Track hazards, closures, risk mitigations

SAE ARP 4761 Civil Avionics Safety Process

Different industries have different guidelines, but they are all founded on common system safety concepts.

FHA - Functional Hazard Assessment
PSSA - Preliminary System Safety Assessment
SSA - System Safety Assessment
MA - Markov Analysis
DD - Dependency Diagrams
ETA - Event Tree Analysis
FTA - Fault Tree Analysis
CCA - Common Cause Analysis
FMEA - Failure Modes & Effects Analysis

$$\begin{tikzcd}
 & & & \text{FHA} \arrow[d] & & \\
 & & & \text{PSSA} \arrow[r] \arrow[dr] & \text{System Design} & \\
 & & & & \text{System Architecture} & \\
 & & & \text{SSA} \arrow[uu] & & \\
\text{MA} \arrow[urrr] & \text{DD} \arrow[urr] & \text{ETA} \arrow[ur] & \text{FTA} \arrow[u] & \text{CCA} \arrow[ul] & \text{FMEA} \arrow[ull]
\end{tikzcd}$$

Some Common Causes of Hazards

- Hardware failures
- Design defects
- Operator error
- Malicious misuse
- Unanticipated circumstances

A number of more detailed guidelines are available.

Risk Perception and Acceptability

Risk perception studies suggest acceptability is influenced by factors other than quantitative probability of injury, death, or damage, e.g.

- Beneficial or non-beneficial
- Voluntarily or involuntarily assumed
- Easy or difficult to imagine
- Familiar or novel
- Socially stigmatized or acceptable

Identifying, assessing and mitigating risks is a system developer responsibility.

Determining whether a risk is acceptable or not is a customer or societal responsibility.

The system safety program defines the process used for informed acceptance of residual risk, for determining what is acceptable or not.

Comment

Quantitative metrics and analyses of various kinds are used for various purposes.

Many things cannot be analytically quantified to a generally accepted level of assurance, there are no adequately validated models.

Qualitative metrics, guidelines, and policies are widely-used of necessity.

Outline

AADL Domain of Applicability

AADL focuses on embedded computing. Standard semantics, properties, etc. are defined for

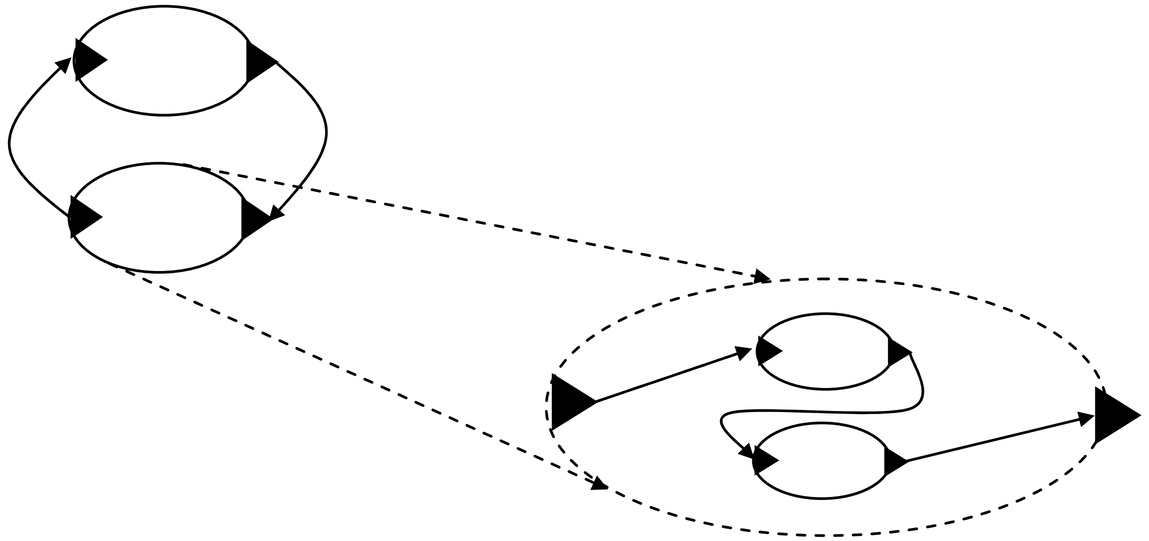
- systems of software and hardware
- resource allocation, scheduling and timing
- faults and errors
- partitioning and security
- common real-time and fault-tolerance design idioms

AADL focuses on architectures. Internal component details such as algorithms, data structures, etc. are specified using other languages such as C, Ada, SimuLink, VHDL, ...

AADL is a relatively permissive standard. The language includes several extension mechanisms, and the standard gives several permissions for super-setting and sub-setting.

Systems of Components

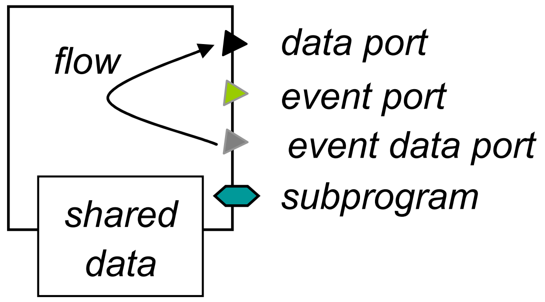
- A *system* is a set of connected *components*.
- A component has a declared *component type*, an interface that defines the possible *connections* between it and other components.
- A component may itself be a system (hierarchical specification is supported).



Component Types

A *component type* must be declared to have one of a set of standard *component categories* that associate standard semantics with components having that type.

data	*device*
subprogram	*memory*
thread	*bus*
thread group	*processor*
process	*system*



A component type may contain declarations of:

- data and event ports
- shareable/callable subcomponents
- flows (internal dependencies between inputs and outputs)
- modes (alternative run-time behaviors or configurations)
- properties (externally visible)

Component Implementations

Zero or more component implementations may be declared for a *component type*.

A *component implementation* may contain declarations of:

- subcomponents (type, implementation, properties)
- connections (between subcomponent ports)
- flows (internal logical dependencies between inputs and outputs)
- properties
- modes (other declarations have mode-dependent forms)
- mode transitions

Interactions between components are declared using

- *connections* between ports
- *binding* properties

of which there are several standard kinds and semantics

AADL Error Model Type

fault and and repair events

error model **Basic**

features

  Fail_Stop, Fail_Babbling : error event**;**

internal error states,
system hazards

  Error_Free: initial error state**;**

  Stopped, Babbling: error state;

external failure modes/effects,
mishaps

  No_Data, Bad_Data : in out error propagation**;**

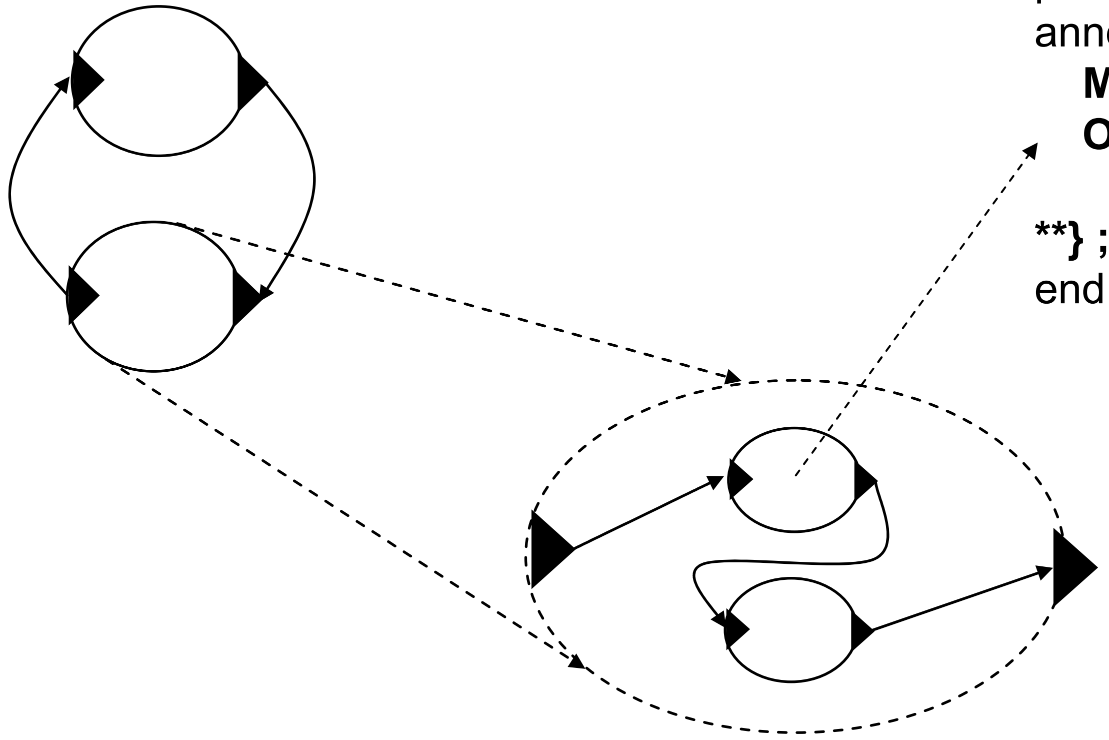
end **Basic;**

AADL Error Model Implementation

```
error model implementation Basic.Nominal
transitions
    Error_Free -[Fail_Stop, in No_Data]-> Stopped;
    Error_Free -[Fail_Babbling, in Bad_Data]-> Babbling;
    Stopped –[ out No_Data ]-> Stopped;
    Babbling –[ out Bad_Data ]-> Babbling;
properties
    Occurrence => poisson 10E-4 applies to Fail_Stop;
    Occurrence => poisson 10E-6 applies to Babbling;
end Basic.Nominal;
```

An error model implementation is a kind of stochastic automaton.
Concurrent stochastic automata can be composed.

AADL Component Error Model



```
processor PPC.M703
annex Error_Model {**
    Model => My_Models::Basic.Nominal;
    Occurrence => poisson 10E-4
                          applies to error Fail;
**} ;
end PPC.M703;
```

System error models are compositions of component error models.

AADL Hierarchical Error Modeling

A subsystem of components may have an explicitly associated error model, in addition to error models for the components.

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.

Voting AADL Error Propagations

```
process A
features
    Sensor_1: in port;
   Sensor_2: in port;
end A;

process implementation A.Nominal
annex Error_Model {**
   Guard_In =>
       mask when (Sensor_1 and Sensor_2[No_Data])
                    or (Sensor_1[No_Data] and Sensor_2)),
       No_Data when (Sensor_1[No_Data] and Sensor_2[No_Data])
                       or (Sensor_1 and Sensor_2[Bad_Data])
                       or (Sensor_1[Bad_Data] and Sensor_2),
       Bad_Data when Sensor_1[Bad_Data] or Sensor_2[Bad_Data]
   applies to Sensor_1, Sensor_2;
**} ;
end A.Nominal;
```

Propagations 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

Report Property

```
Report => Stopped, Babbling;
```

The Report property association declares a set of error states of interest for analysis, e.g. the probability that the associated component will be in a named error state is to be included in the analysis report.

Safety Analysis Requirements

Support product families, openness, upgradeability

- Compositional

Use early in development for architecture optimization

- Fast generation and analysis from partial specifications
  - incremental
  - abstract
  - numerically tractable
- Results traceable back to architecture alternatives
  - tree structure maps clearly to architecture structure
  - parametric analyses
  - scenario analysis

Use late in development for verification/certification

- Verifiably traceable to requirements and implementation
  - verifiable that tree structure maps to architecture specification
  - verifiable that architecture maps to requirements and implementation
- Mimic current safety processes

Series of Prototype Tools

Stochastic Concurrent Automata and Markov Chains

- General (cyclic) component error models
- Prototype generator targeting SURE, Mobius

Fault Trees

- Cycle-free component error models
- First prototype in MetaH targeting Item Toolkit
- Second prototype in OSATE/TOPCASED targeting CAFTA

Why Fault Trees?

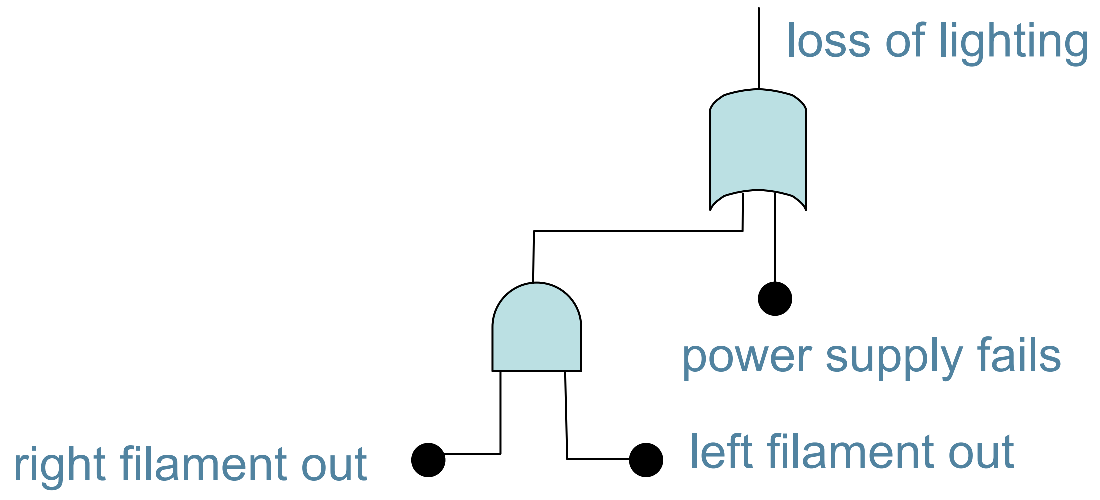
More tractable for industrial-scale problems.

Widely-accepted and used.

Outline

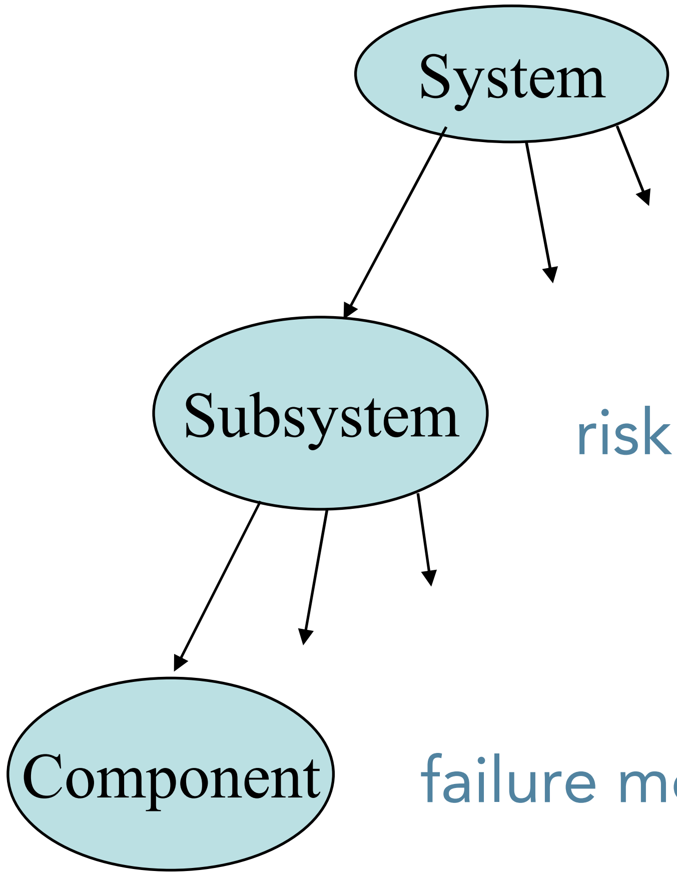
Fault Trees

A fault tree looks like a Boolean circuit, with a few higher-level operators convenient for this domain (e.g. N ORMORE).



Given a set of basic events that are True, a True/False value can be computed for the root.

Top-Down and Bottom-Up



system hazards

risk mitigation architecture

failure modes & effects analysis (FMEA)

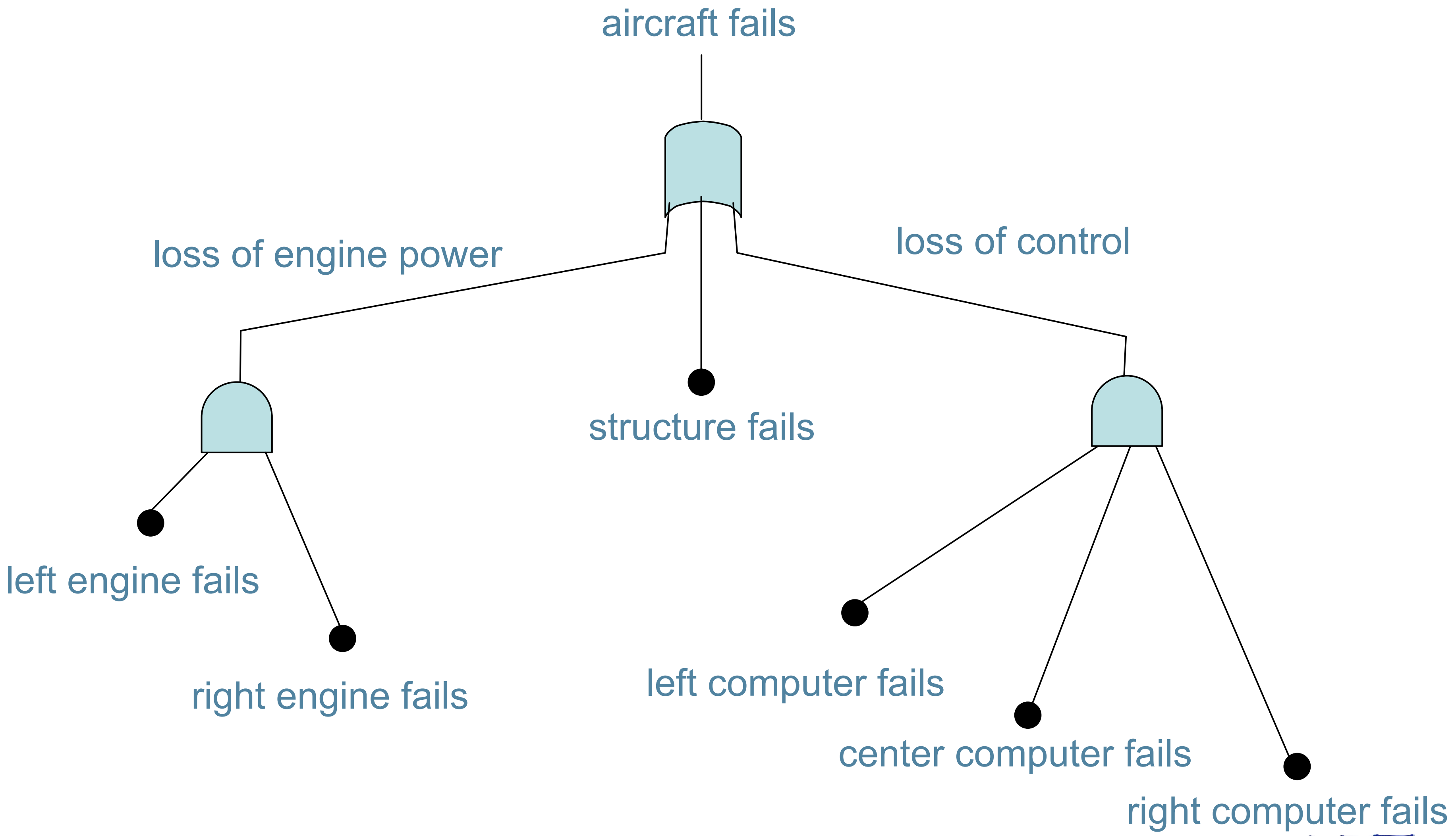
Fault Trees

Fault trees can relate system hazards (fault tree roots) to component failure modes and effects (fault tree inputs).

Fault tree structure models the risk mitigations, e.g. redundancy.

Fault tree structure captures relevant aspects of the architecture structure and behavior.

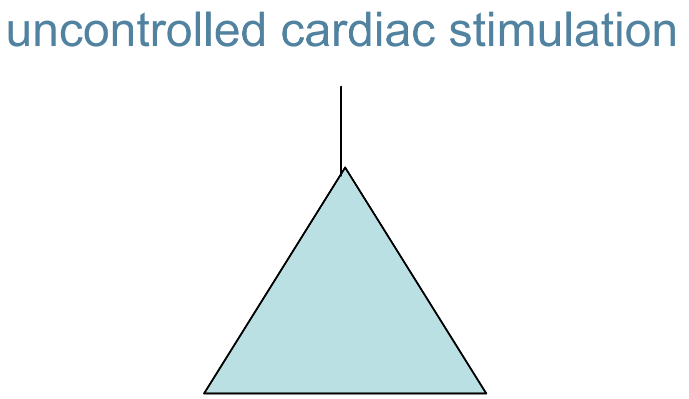
Trivial Example



One Root per System Hazard

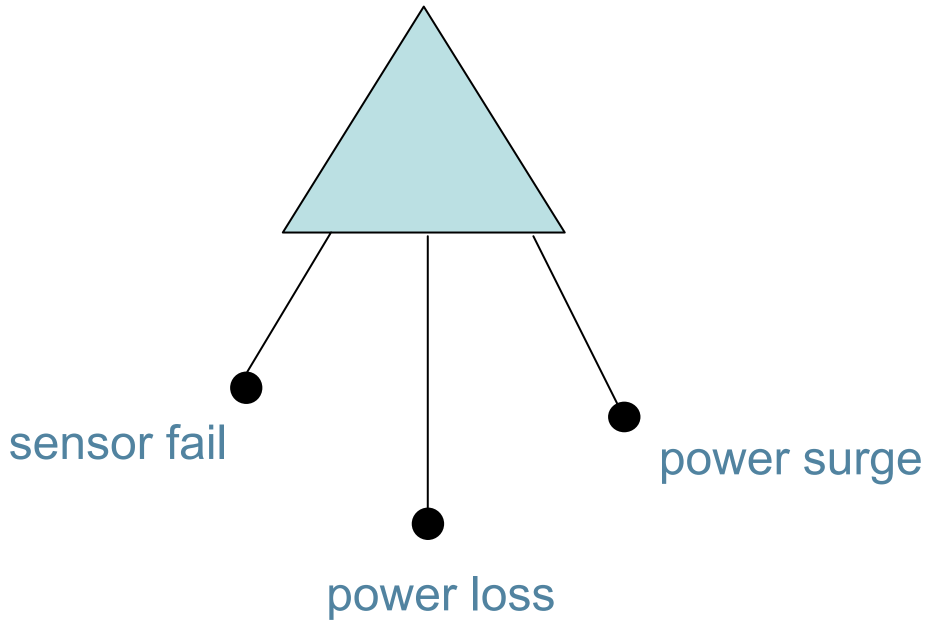
The root (output) of the tree specifies the conditions under which the system will be in a hazardous state.

There is typically one fault tree for each identified system hazard (a fault forest or DAG).



One Basic Event per Failure

The basic events (leaves) identify the significant component failure modes or effects.

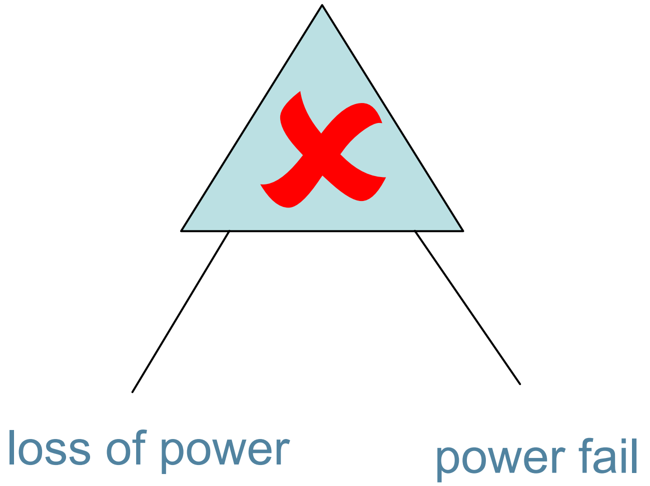


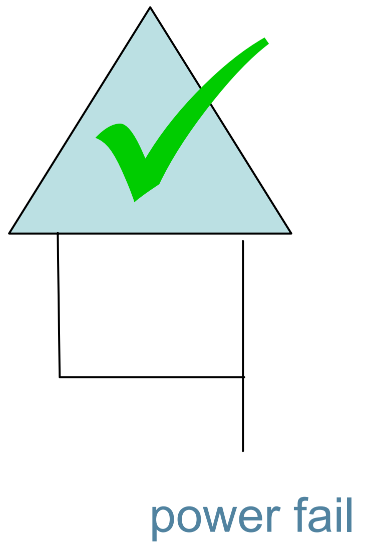
Basic Events Are Assumed Independent

Basic events are assumed to be (statistically) independent.

Any common cause basic events must be clear in the fault tree.

Tools have different conventions for this, e.g. may look at name equivalence.





A Discrete Fault Tree Analysis

Cut sets are the shortest lists of combinations of basic events that can cause tree failure.

- Are there any single events that can cause failure?
- Are there any pairs of events that can cause failure?
- ….

A Stochastic Fault Tree Analysis

A basic event is assigned a probability of being True, e.g. a probability of occurring in a specified interval of time.

Calculate the probability the root will be True, e.g. the probability of that system hazard in that interval of time.

Importance and parametric sensitivity analyses are possible, e.g. what basic event probability contributes most to the root probability of failure?

More complex semantics are possible, e.g. basic events have recovery as well as failure probabilities and may toggle True/False.

Outline

System Safety

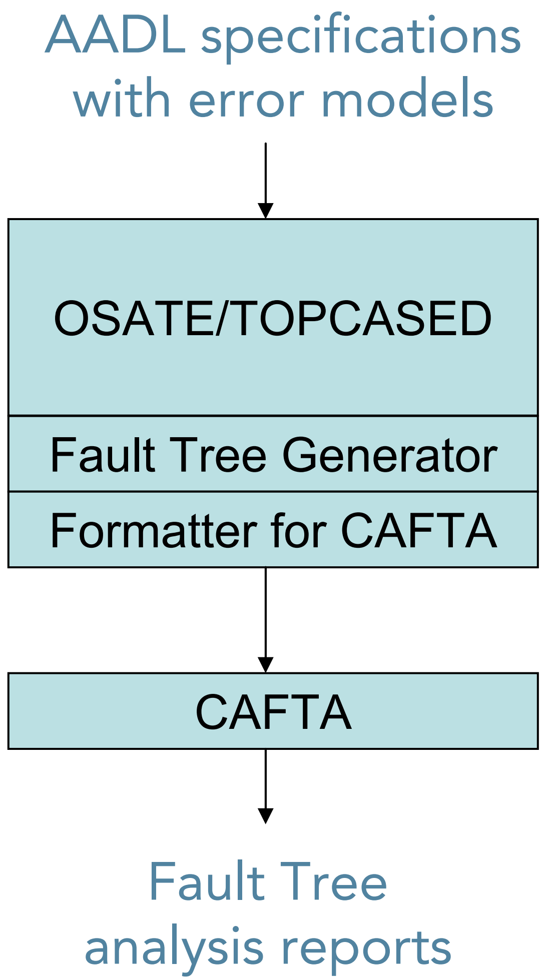
AADL Architecture and Error Modeling

Fault Trees

➡ Generating Fault Trees from AADL Specifications

Referrals

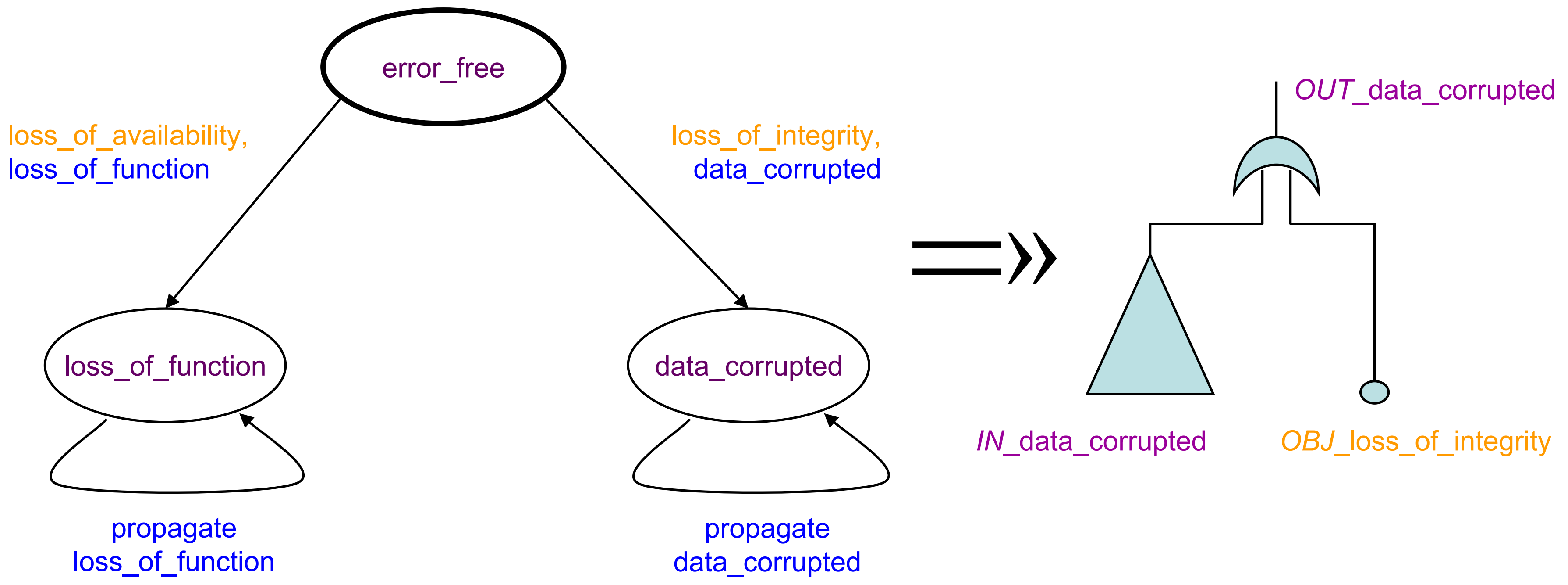
Most Recent Prototype Toolset Architecture


AADL specifications with error models
OSATE/TOPCASED
Fault Tree Generator
Formatter for CAFTA
CAFTA
Fault Tree analysis reports

High-Level Algorithm

- Generate top-down from each Report (hazard) analysis requirement.
- Compose a fault tree template constructed from a component’s error model with its input sub-trees, each obtained from a potential source of errors for that component.
  - internal errors (basic events in the fault tree template)
  - propagated errors (inputs to the fault tree template)
- Common basic events and sub-trees must be detected.
- Cycles in the architecture specification must be broken.
- Optimizing transformations should be applied.

Translating Error Models to Fault Tree Templates



One fault tree template is constructed for each propagate-able error state.

A reported hazard fault tree is built by **composing** fault tree templates.

Restricted to cycle-free error models (cyclic models require dynamic analysis, e.g. dynamic fault trees, Markov).

Experience with First MetaH Prototype

Specification extracted from an anticipated civil IMA system.

- 8 common processors, 18 IO processors
- 12 switches, 62 physical links
- 40 hosted functions with about 1300 message channels
- assumed a dual-redundancy pattern with fail-stop partitions

Generated two fault trees per function

- loss of availability
- loss of integrity

Fault tree sizes ranged from about 10 to 2500 gates

All trees generated in 30 seconds (incremental supported)

Largest tree analyzed in 10 minutes

Future Fault Tree Generator Work

Consistent, complete, reviewed mathematical basis for generating trees that are optimized, intuitive, and traceable

Incrementally manage forests of DAGs

Further validation against real-world complexity

- more than two system hazards per function
- more complex FMEA models
- many redundancy patterns and consensus protocols

Multi-function analysis

Dynamic analysis

Consistency of analytic models, simulations, implementations (e.g. sound fault injection experiments)

Cultural, process, certification concerns about a novel method

Outline

System Safety

AADL Architecture and Error Modeling

Fault Trees

Generating Fault Trees from AADL Specifications

➡ Referrals

Antoine Rauzy, “A New Methodology to Handle Boolean Models With Loops,” IEEE Transactions on Reliability, March 2003.

Anjali Joshi, Steve Vestal, Pam Binns, “Automatic Generation of Static Fault Trees,” *DSN Workshop on Architecting Dependable Systems, 2007.*

Hongyu Sun, Miriam Hauptman, Robyn Lutz, “Integrating Product-Line Fault Tree Analysis into AADL Models,” *High Assurance Systems Engineering Symposium, 2007.*

Ana-Elena, Rugina, Karama Kanoun, Mohamed Kaâniche, “The ADAPT Tool: From AADL Architectural Models to Stochastic Petri Nets through Model Transformation,” *Seventh European Dependable Computing Conference, 2008.*

Pierre Bieber, Christian Bougnol, Charles Castel, Jean-Pierre Heckmann, Christophe Kehren, Sylvain Metge, Christel Seguin, “Safety Assessment with AltaRica,” *Building the Information Society*, Springer Verlag, 2008.

Laboratoire d’Analyse et d’Architecture des Systemes (LAAS) Dependable Computing and Fault Tolerance, http://www.laas.fr/laas/2-4287-TSF.php

University of York High Integrity Systems Engineering, http://www.cs.york.ac.uk/research/hise.htm

University of Illinois Center for Reliable and High-Performance Computing, http://www.crhc.uiuc.edu/