Fault Tree Analysis of UML Designs

Dr. Christopher Harper

Director, Avian Technologies Ltd.
www.avian-technologies.co.uk
cjharper@avian-technologies.co.uk

Alan Parkinson

Director, AGP Micro Ltd.
www.agpmicro.co.uk
alan@agpmicro.co.uk

© 2007 Avian Technologies Ltd

Contents

- The Engineering Problem
- Failure Analysis using Fault Trees
- UML Semantics
- Fault Tree Analysis of UML
- Model-driven Safety Validation
- Current work

© 2007 Avian Technologies Ltd

AVIAN TECHNOLOGIES
SAFETY CRITICAL & INTELLIGENT SYSTEMS ENGINEERING

The Critical Systems Engineering Problem

SW Safety Requirements
System Requirements
Safety Standards
Validated Software
Safety Case Preparation
Software Requirements
Design Validation Test Spec
Software Validation
Software Architecture
Derived Requirements
Software Architecture
Verified Software
Safety Validation Test Specification
Software Hazard Analysis
- HAZOP/FFA

Safety Validation
- Safety-directed acceptance testing

Software Design (UML)
Design Verification Test Spec
Software Verification
Software Design
Derived Requirements
Safety Verification Test Spec
Software Safety Analysis
- FTA, FMEA

Safety Verification
- Safety-directed unit / component testing

Software Design
Source Code
Implementation
Source Code
Code Changes
Implementation Safety Assessment
- Design review

- Critical systems require dependability assurance
- Dependability assurance requires a lot of extra work (analysis, review, testing)
- The extra work can introduce late design changes into the system
- Late design changes are extremely expensive!

© 2007 Avian Technologies Ltd

Train Brake Control System: Context Use Case Diagram

Train Brake Control System

Emergency Braking

Wheel Slide Protection

<<include>>

<<include>>

Service Braking

<<extend>>

Brake Management

Door Control

Emergency Brake Button

Driver Service Brake Handle

Low Pressure Sensor

Fire Alarm Signal

Door Control Relay

Brakes

Brake Cylinder Pressure

Tacho Sensors

© 2007 Avian Technologies Ltd

AVIAN TECHNOLOGIES
SAFETY CRITICAL & INTELLIGENT SYSTEMS ENGINEERING

Door Control: Class Design

Class Model

© 2007 Avian Technologies Ltd

AVIAN TECHNOLOGIES
SAFETY CRITICAL & INTELLIGENT SYSTEMS ENGINEERING

Train Brake System: Interaction Diagram

DoorController

WheelTacho

FireAlarmSignal

Door Control Relay

ControlDoors()

getAxleSpeed()

Speed AxleSpeed

IsFireAlarmActive()

bool AlarmActive

alt

[AlarmActive = true & AxleSpeed < 5]

OpenDoors

[else]

LockDoors

© 2007 Avian Technologies Ltd

Example Fault Tree

Doors open while train is moving (>= 5kph)

OpenDoors signal sent when not intended

LockDoors signal not sent when intended

Not developed in this example

AxleSpeed < 5 when not intended

AlarmActive = true when not intended

getAxleSpeed() returns incorrect value of AxleSpeed

Internal errors within operation algorithm

Tacho sensor input reads < 5kph when not intended

Tacho sensor fault

Wheel slip causes low speed measurement

getAxleSpeed() called when not intended

ControlDoors() calls getAxleSpeed() when not intended

Natural part of algorithm !!

ControlDoors() called when not intended

Commission error of ControlDoors()

FireAlarmState attribute set Active when not intended

Fire sensor hardware faults

IsFireAlarmActive() called when not intended

ControlDoors() calls IsFireAlarmAcvtive() when not intended

Natural part of algorithm !!

ControlDoors() called when not intended

Commission error of ControlDoors()

IsFireAlarmActive() returns incorrect value of FireAlarmState

Internal errors within operation algorithm

© 2007 Avian Technologies Ltd

UML 2.0+ Semantics (1)

- UML semantic model has two parts, the semantics architecture and the causality model [Selic]

Semantics Architecture:

Activities	State Machines	Interactions

Behavioural Base

- behaviours
- behavioural classifiers

Actions

- Defines semantics of individual actions
- Communication, Read/Write, Computation, Structural Actions

Inter-object Behaviour Base

- Defines how structural entities communicate
- Sync/async calls, operations/receptions, events, triggers

Intra-object Behaviour Base

- Defines how structural entities behave internally
- Activities, structured activities, nodes, edges, flows

Structural Foundations

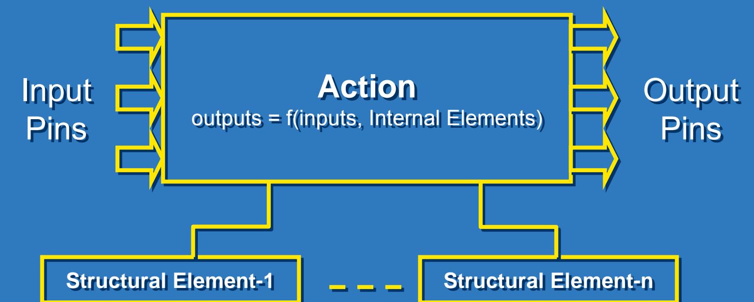
- no disembodied behaviour in UML
- all behaviour generated by actions of structural elements
- pure values, variables, objects, links, messages, composites/aggregates

© 2007 Avian Technologies Ltd

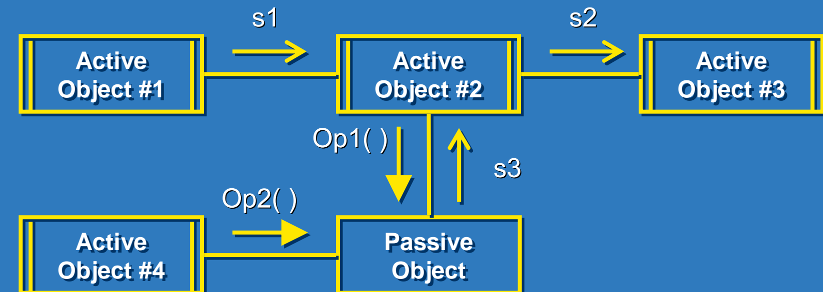
AVIAN TECHNOLOGIES
SAFETY CRITICAL & INTELLIGENT SYSTEMS ENGINEERING

UML 2.0 Semantics (2)

- Basic unit of behaviour is the action
- Actions are specified as:
  - Output Pins = f(Input Pins, Structural Elements)
- Pins are the parameters of class operation signatures



- Causality Model [Selic]
  - Active objects have a thread
    - → time of response not dictated by calling object
    - → run-to-completion semantics
    - → messages arriving between activations are buffered
  - Passive objects run only when called
    - → responds regardless of whether a previous call has been completed (requires mutex to protect, if necessary)



© 2007 Avian Technologies Ltd

Semantic Variation Points

- Not all the semantics required to specify correct operation is defined within the UML standard, BUT…
- 'Placeholders' are left for the gaps – Variation Points
- Variation points must be completed by suppliers of executive software systems, e.g. RTOS, Comms
- Care must be taken! Models may have different behaviour under different variation points. <u>Assumptions about semantics must be recorded</u>.
- But, given these caveats, we believe that the semantics of UML is (just) sufficient to support more advanced model-based analysis, (e.g. failure analysis)

© 2007 Avian Technologies Ltd

Failure Modelling

- Failure analysis is only complete with respect to a particular *failure model*, which must be a prior assumption at the start of any analysis
- Failure model used in this methodology is drawn from techniques developed by David Pumfrey at the University of York [Pumfrey]
- Six basic software failure types:

– Service *Provision* Failures:	Omission, Commission
– Service *Timing* Failures:	Early, Late
– Service *Value* Failures:	Coarse (implausible value), Subtle (plausible value)

- Failure model used extensively by UoY in conjunction with aerospace industry (e.g. Airbus, BAE Systems) and by Avian Technologies on contract work in the rail sector

© 2007 Avian Technologies Ltd

AVIAN TECHNOLOGIES
SAFETY CRITICAL & INTELLIGENT SYSTEMS ENGINEERING

UML Interpretation of Failure Model

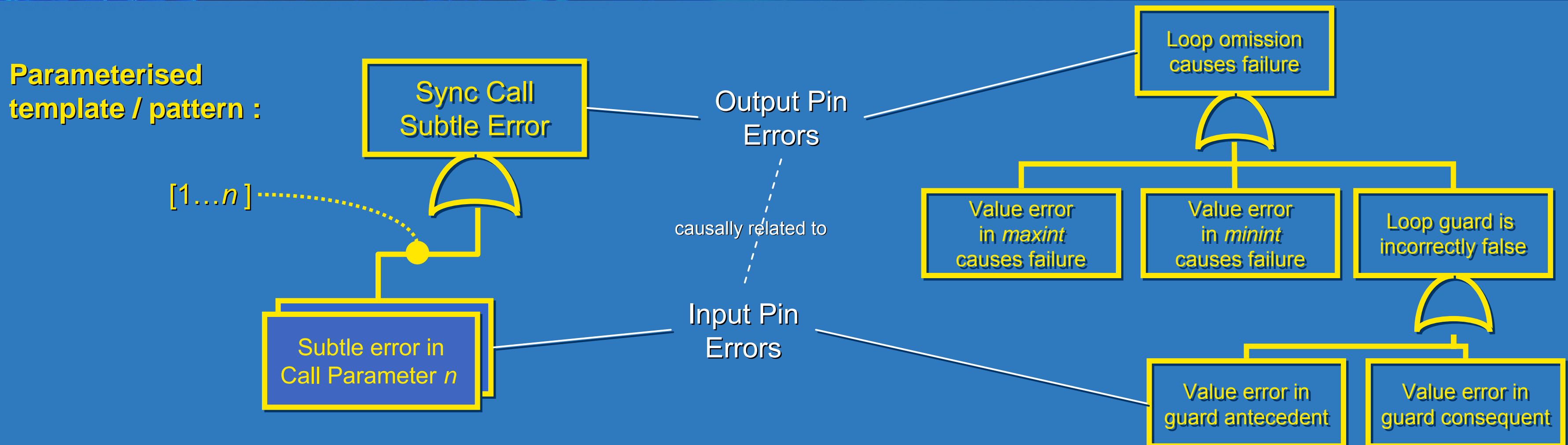
- The standard failure types must be *interpreted* in UML terms:

UML Model Element Type	Service Provision Error		Service Timing Error		Service Value Error	
	Omission	Commission	Early	Late	Coarse (Implausible)	Subtle (Plausible)
Synchronous Operation Call	Call Action associated with message is not executed at the intended point in the behaviour	Call Action associated with this message executed instead the intended one	N/A (GeneralOrdering error?)	N/A (GeneralOrdering error?)	Implausible values taken by operation parameters (if possible)	Plausible, but incorrect, values taken by operation parameters
Combined Fragment : Loop	Loop body not executed when intended: – *minint* parameter value error (is too low, or is zero) – *maxint* parameter value error (too low) – guard condition incorrectly false	Loop body executed when not intended: – *minint* parameter too high – *maxint* parameter too high (or infinite) – guard condition incorrectly true	As commission	As omission	*N/A*	N/A

© 2007 Avian Technologies Ltd

AVIAN TECHNOLOGIES
SAFETY CRITICAL & INTELLIGENT SYSTEMS ENGINEERING

UML Fault Tree Construction Methodology

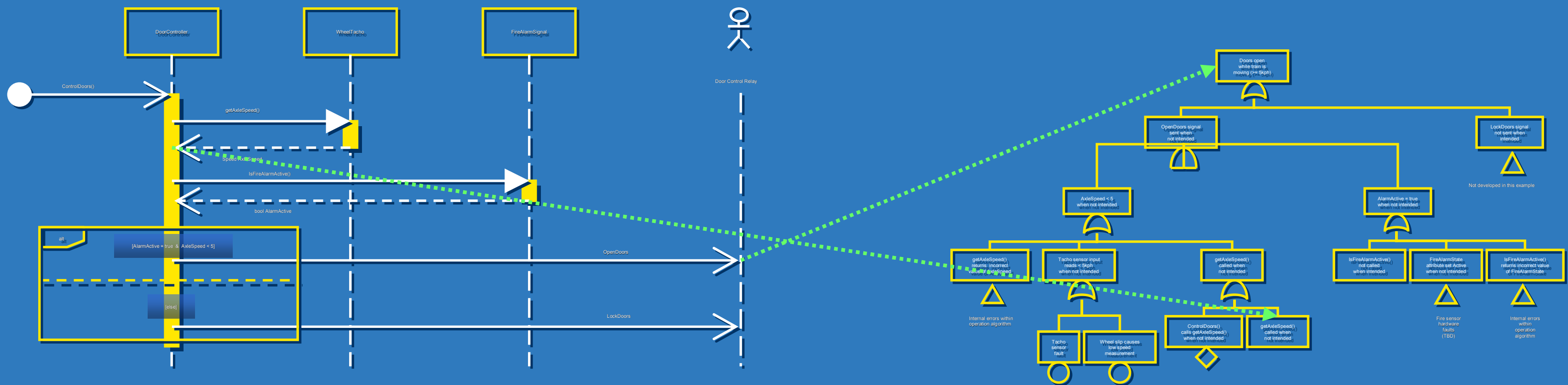


- A fragmentary fault tree template is defined for every type of failure of every type of model element in a class of UML behaviour diagram [cf. Leveson et al]. Each fragment relates the failure type of the associated UML Action (at its output pins) to a set of failures of its input pins.
- The causal logic of each fragment is specific to the type of action. Its structure may depend on semantic variation points – in this case the relevant variation must be selected as the analysis unfolds
- Pattern-based methodology (cf. OO design patterns, safety case patterns [Habli & Kelly])

© 2007 Avian Technologies Ltd

AVIAN TECHNOLOGIES
SAFETY CRITICAL & INTELLIGENT SYSTEMS ENGINEERING

Systematic Integration of Analyses and Models



- The template-driven approach ensures that a consistent interpretation is taken of the causal relationships between input and output pins of actions
- A UML diagram can be scanned automatically to obtain the overall structure of its associated fault tree
- The end result is a Fault Tree that is logically consistent with, and systematically derivable from, its associated UML Behaviour Diagram.
- The procedure can be (partially) mechanized, thereby accelerating the analysis process and improving its consistency
(but a human analyst must still decide which template fits into each slot)

© 2007 Avian Technologies Ltd

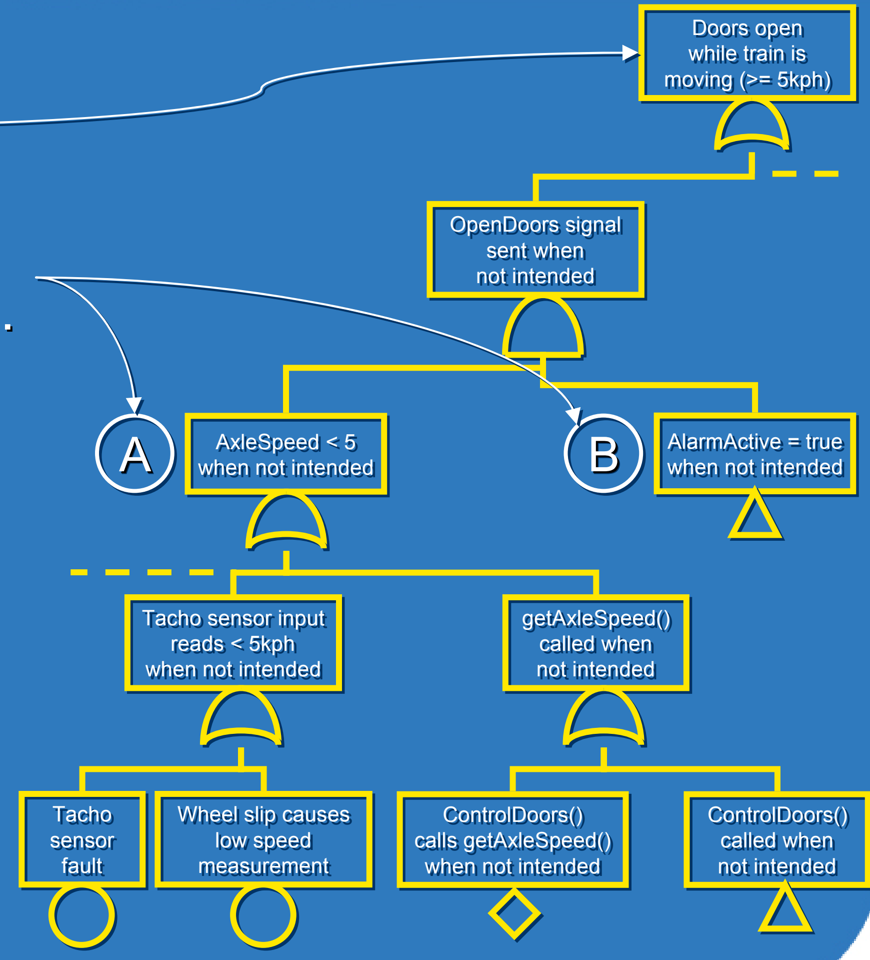
AVIAN TECHNOLOGIES
SAFETY CRITICAL & INTELLIGENT SYSTEMS ENGINEERING

The Counterfactual Nature of Failure Analysis

IF this system failure occurs…

THEN these are the software errors that would have to have happened…

and these are the basic faults that would have to have happened to cause error A …



© 2007 Avian Technologies Ltd

AVIAN TECHNOLOGIES
SAFETY CRITICAL & INTELLIGENT SYSTEMS ENGINEERING

The Counterfactual Nature of Failure Analysis (2)

BUT :
if the basic faults
don't happen …

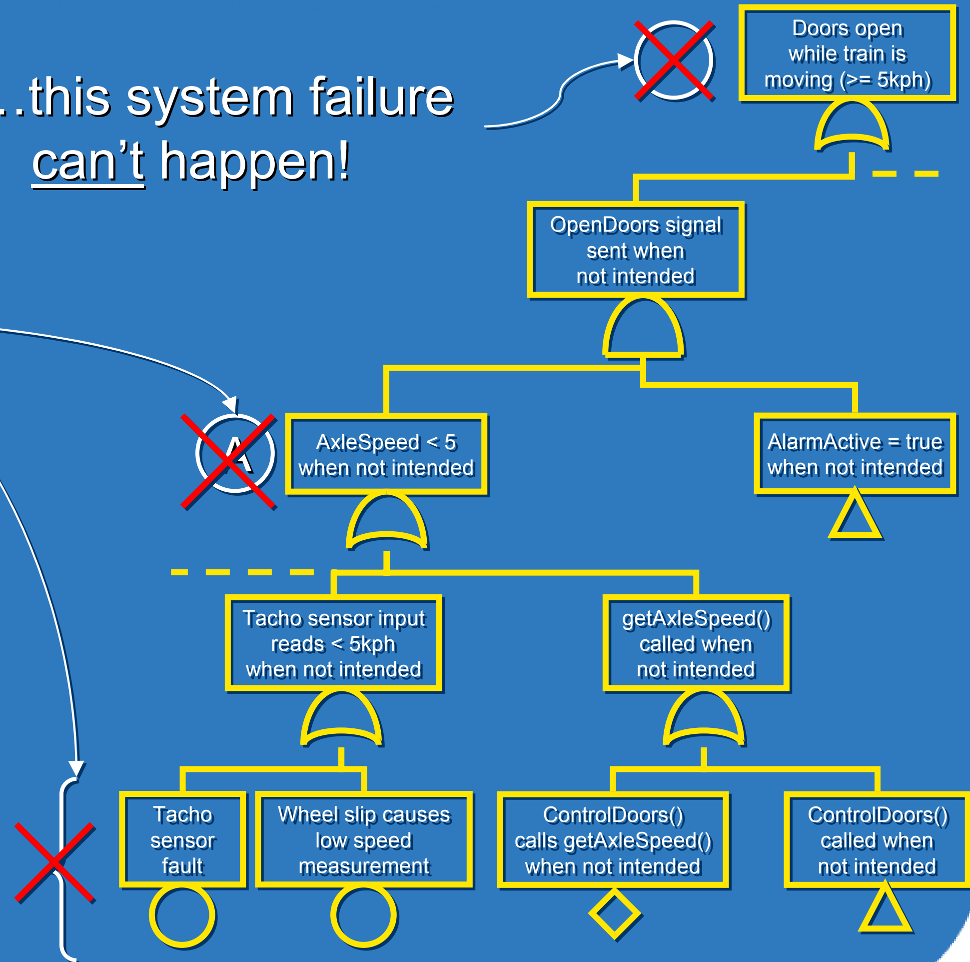
…this system failure
can't happen!

SO :

If we test the software for the existence of the basic faults…

…we can determine whether the software will cause the system-level hazard.

This is the real value of performing a software FTA.



© 2007 Avian Technologies Ltd

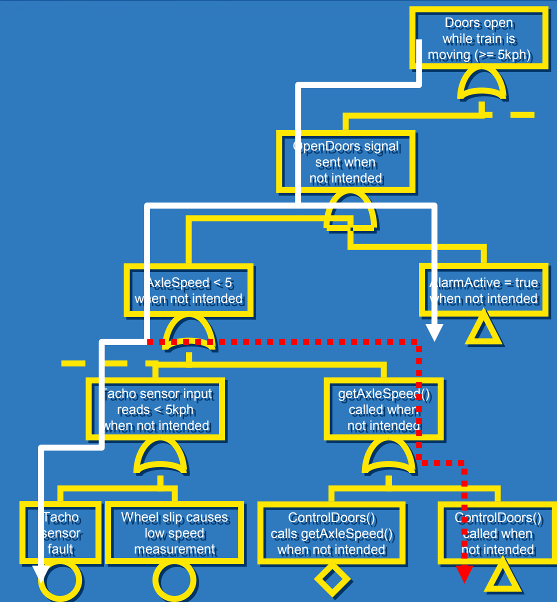
Synthesis of Safety Validation Tests from Fault Trees

A fault tree corresponds with a set of DNF (sum-of-products) statements, which are derived from the paths from the top condition to the base events. Two statements are shown below, derived from the paths shown by the white trace through the tree:

1. (Tacho_fault $\wedge$ FireAlarmState=Active)
2. (Wheel_slip $\wedge$ FireAlarmState=Active)

These sets of events are, in effect, test patterns that cause the potential hazard to occur at the output. They can be applied as test cases to verify this fact.

Fault trees can also reveal test cases for other operations/classes/modules that require assessment, because they are sources of sensitive base events causing the failure condition (e.g. red dotted path).



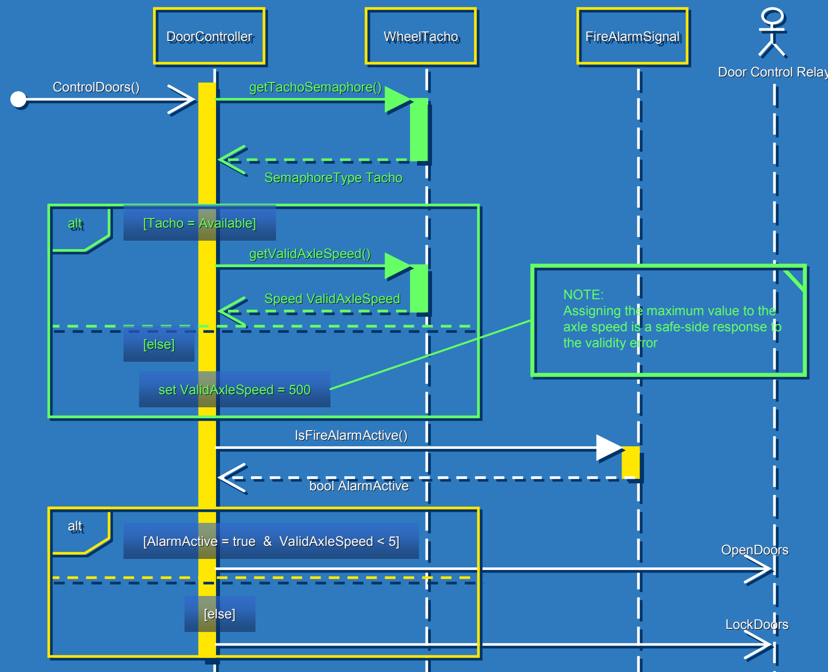
- Test cases can be derived easily from fault trees, because fault trees specify a set of *weakest pre-conditions* for the occurrence of a hazard (the top condition of the tree) [McDermid&Clarke].
- The inverse or negation of each pre-condition forms a verification condition, or could be re-written as a design inspection criterion.
- UML is well-suited to this because it includes the language OCL, which is intended for the specification of design pre-condition and post-condition information.

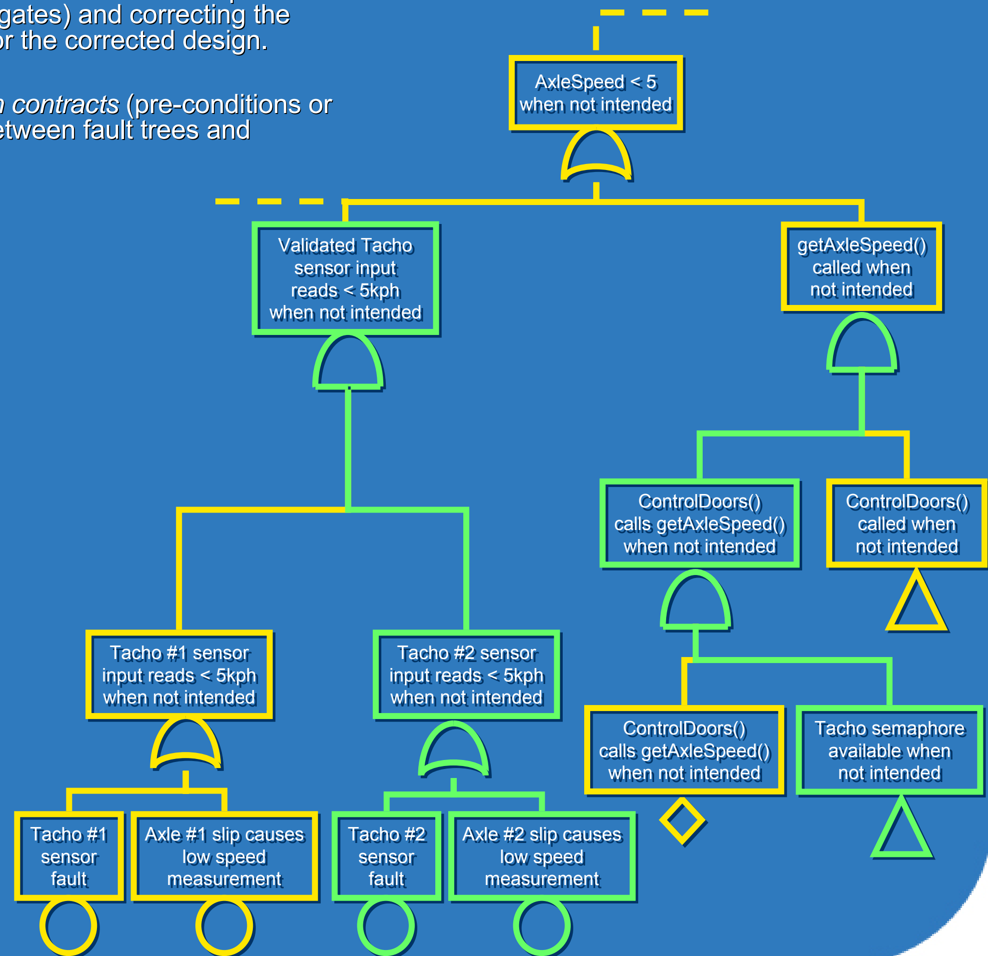
© 2007 Avian Technologies Ltd

AVIAN TECHNOLOGIES
SAFETY CRITICAL & INTELLIGENT SYSTEMS ENGINEERING

Synthesis of Design Corrections from Fault Trees

- Fault trees can be used to drive design changes by exploring the tree for vulnerable points (where paths from top events to base events consist only of OR-gates) and correcting the design so as to (re)introduce AND gates into the new fault tree for the corrected design. [Illustrated by changes marked in green.]
- This can also be done by the creation and enforcement of *design contracts* (pre-conditions or post-conditions to code execution) because of the relationship between fault trees and weakest preconditions.





© 2007 Avian Technologies Ltd

AVIAN TECHNOLOGIES
SAFETY CRITICAL & INTELLIGENT SYSTEMS ENGINEERING

HiCASE – A New Software Tool for Safety Assurance of SysML/UML Designs

Avian Technologies and AGP Micro are working on a joint venture to develop HiCASE - a software tool for safety assurance and engineering of UML (and eventually SysML) system designs:

- Objectives
  - Reduce costs of developing high integrity systems/software by improving integration of safety processes with design processes
  - Semi-automatic generation of safety analyses directly from UML/SysML models
  - Automatic generation of safety validation tests/checklists from causal analyses (fault trees)
  - Automatic generation of safety properties as class contracts for design-by-contract processes and formal verification (e.g. SPARK)
- Production Schedule (approximate)
  - Product launch early summer 2007
  - Functionality upgrades/enhancements at least once per year (for subscription holders)
- Product Versions
  - Professional Edition: basic UML model scanning and safety analysis capabilities only
  - Enterprise Edition: full range of functions
  - Certification Pack: additional documentation for DefStan00-56 qualification (all SILs)

© 2007 Avian Technologies Ltd

References

1. Anda B., Hansen K., Gullesen I, Thorsen H.K., Experiences from Introducing UML-based Development in a Large safety-Critical Project, URL: http://www.simula.no/departments/engineering/publications/Anda.2005.3
2. Pataricza A., Majzik I., Huszerl G., Várnai Gy.: *UML-based Design and Formal Analysis of a Safety-Critical Railway Control Software Module,* Proc. Symp. FORMS-2003, Budapest, May 15-16, pp.125-132, 2003
3. Selic B.V., *On the Semantic Foundations of Standard UML 2.0*, LNCS 3185, pp181-199, Springer Verlag 2004
4. Leveson N.G., Cha S.S, Shimeall T.J., *Safety Verification of Ada Programs Using Software Fault Trees*, IEEE Software, July 1991, pp48-59
5. McDermid J.A., Clarke S.J., *Software fault trees and weakest preconditions: a comparison and analysis*, IEE Softw. Eng. Journ., July 1993, pp225-236
6. Pumfrey D., *The Principled Design of Computer System Safety* Analyses, PhD Thesis, University of York 1999
7. Habli I., Kelly T., *Achieving Integrated Process and Product Arguments*, Proc. 15th Safety-critical Systems Symposium, Eds. Redmill and Anderson, Springer 2007 (also: the tutorial in this OMG Software Assurance Workshop)

© 2007 Avian Technologies Ltd