Towards Trustworthy Aerospace Systems
using Formal Methods

Joost-Pieter Katoen

Software Modeling and Verification Group
RWTH Aachen University

Grande Region Security and Reliability Day 2013, Luxembourg

joint work with Marco Bozzano, Harold Bruintjes, Alessandro Cimatti, Christian Dehnert, Marie-Aude Esteve, Viet Yen Nguyen, Thomas Noll Xavier Olivé, Bart Postma, Marco Roveri and Yuri Yushstein

Agenda

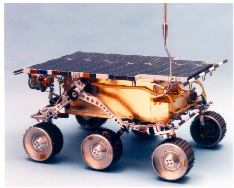
Overview

Aerospace systems

Weather satellite

Ariane 5

Space station ISS



Mars Pathfinder

GPS system with 26 satellites

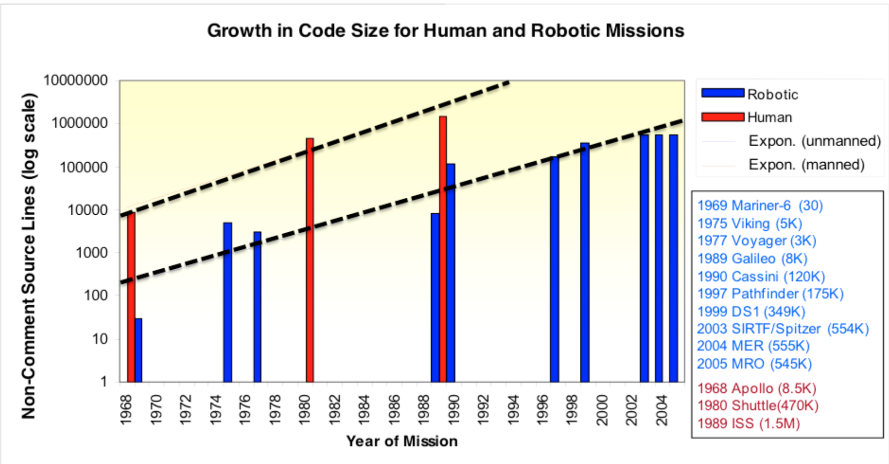
A Lego starwars ship

Another aerospace system

Extreme dependability!

- They must offer service without interruption for a very long time – typically years or decades.
- 'Five nines' dependability is not sufficient.
- Faults are costly and may severely damage reputations, e.g. Ariane 5.

Spacecraft := flying software



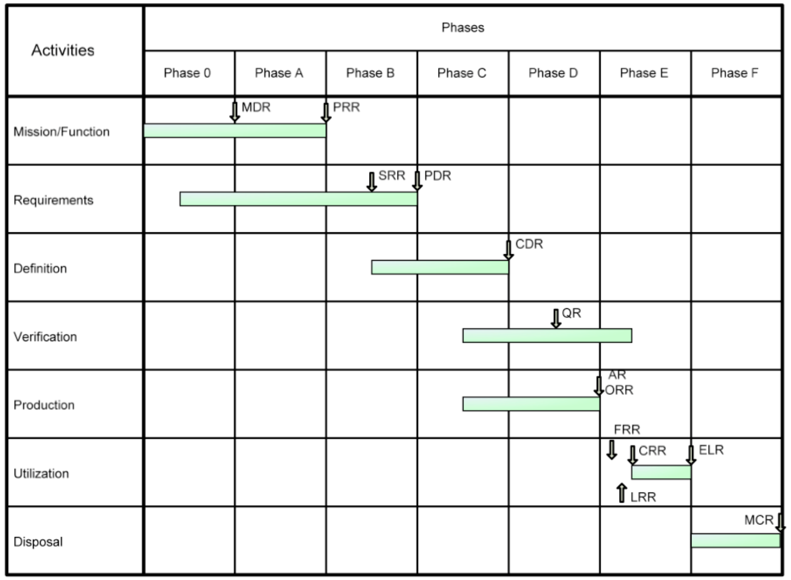
NASA Study Flight Software Complexity (2009)

What do we need?

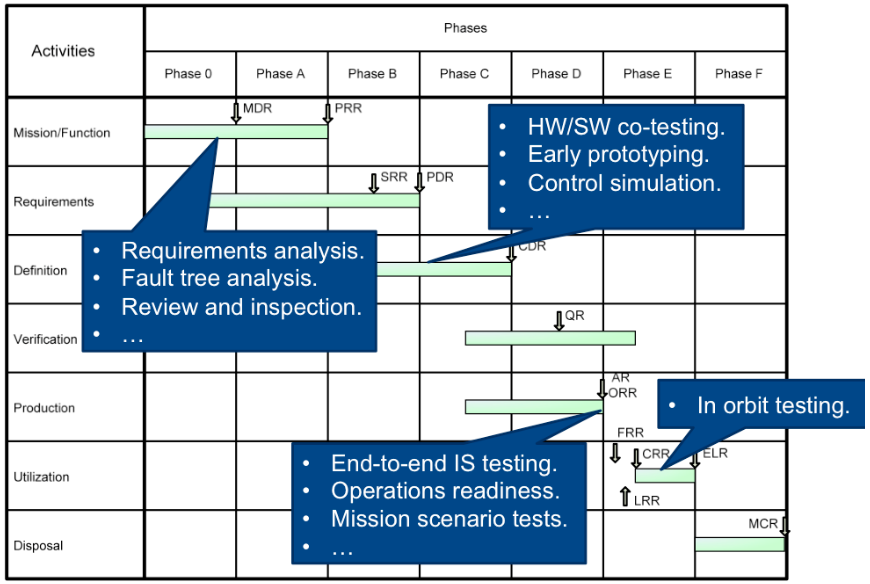
Challenges

- Rigorous design support and analysis techniques are called for.
- Bugs must be found as early as possible in the design process.
- Check performance and reliability guarantees whenever possible.
- The effect of Fault Detection, Isolation and Recovery (FDIR) measures must be quantifiable.

Spacecraft design process



Verification and validation in spacecraft design



Weaknesses and limitations

Software is mostly verified in isolation from the target hardware.

Limited support for modeling fault models and degraded modes of operation.

Distinct modeling formalisms and analysis techniques for different system aspects.

Limited support for checking timed, hybrid, and probabilistic properties.

No coherent approach to study effectiveness of FDIR [1]

[1] Fault Detection, Isolation and Recovery

Our objective

Develop an integrated system-software co-engineering approach to ensure completeness and consistency from heterogeneous specification and analysis techniques.

Main ingredients should be a general-purpose modeling language, accompanied with a plethora of formal analysis techniques and supported by powerful software tools.

Current situation

Yes, “formal methods" are applied to aerospace systems, but not in a coherent manner at the systems engineering level.

COMPASS project partners

Consortium

- RWTH Aachen University
  Software Modeling and Verification
- Fondazione Bruno Kessler
  Embedded Systems Group
- Thales Alenia Space
  World-wide #1 in satellite systems
- Ellidiss
  GUI developer

Financial support + supervisor

- European Space Agency

Approach in a nutshell

Design a modeling language based on (core) AADL and its Error Annex.

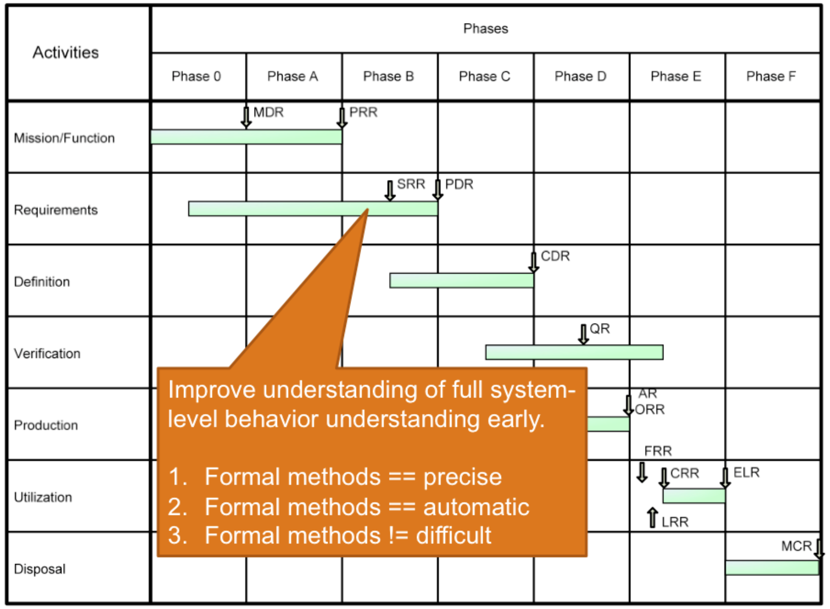
Equip this modeling language with a formal semantics.

Use specification patterns to ease the specification of system properties.

Support the system-engineering language by powerful model-checking tools for correctness, safety, performance and dependability analysis

Evaluate their effectiveness by industrial case studies.

Increase formality in spacecraft design

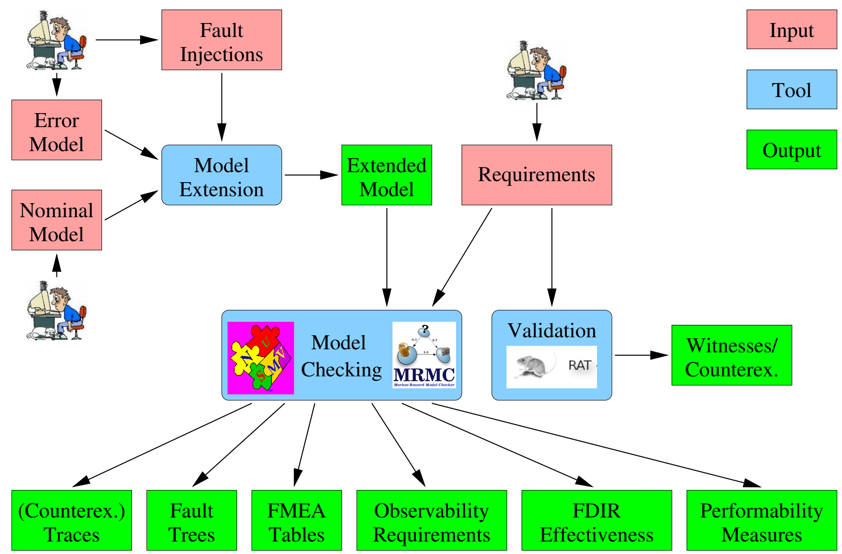


COMPASS phases

1. Project kick-off — February 2008
2. Language design and software tool specification
3. Formal semantics — October 2008
4. Prototype tool implementation — April 2009
5. Prototype evaluation (by industry)
6. Final tool implementation — December 2009
7. Final tool evaluation (by industry) — March 2010
8. Project extension — until March 2011
9. Follow-up projects (NPI, CGM, ESA Case Study) — until September 2012
10. Future projects (HASDEL, D-MILS) — from November 2012

Total budget: $\approx$ 1.3 MEuro; at peak times $\approx$ 10 programmers involved

Approach



Overview

The industry standard AADL

- **1989** MetaH
- **1998** SAE AS-2C
- **2004** AADL 1.0
- **2006** Error Annex 1.0
- **2009** AADL 2.0
- **2010** Error Annex 2.0
- **2014** Error Annex 3.0

Paradigm

- Architecture-based and model-driven top-down and bottom-up engineering
- Real-time and performance critical distributed systems
- Complements component-based product-line development

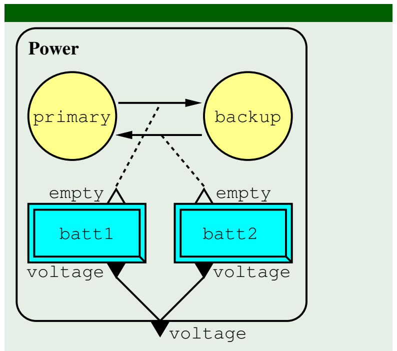
(Our) AADL example: redundant power system

Redundant power system

- Contains two batteries
- Power switches from `primary` to `backup` mode (and back) when `batt1` (`batt2`) is empty

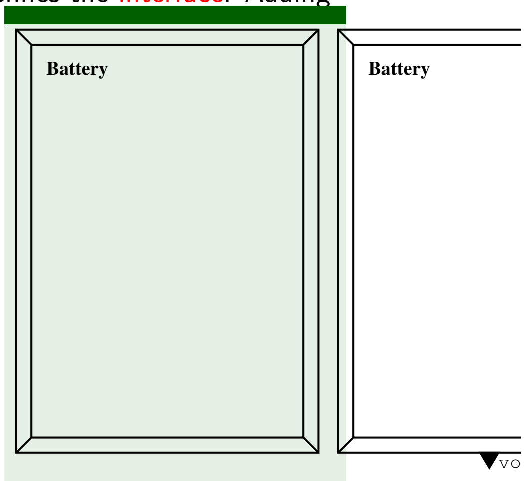
We shall show:

- hybrid behaviour of the batteries
- composition of the power system
- formalisation to automata
- semantics as transition systems
- interweaving of errors



Modeling a battery

Component type and implementation: Type defines the interface: Adding modes behavior: Adding hybrid behavior:

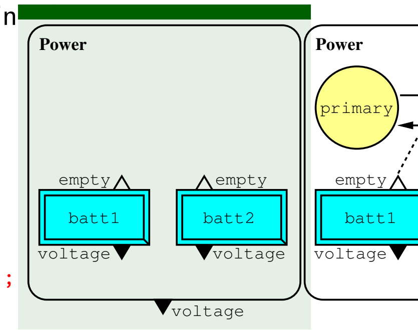



```
device type Battery
  features
    empty: out event port;
    voltage: out data port real default 6.0;
end Battery;

device implementation Battery.Imp
  subcomponents
    energy: data continuous default 100.0;
  modes
    charged: activation mode
      while energy'=-0.02 and energy>=20.0;
    depleted: mode
      while energy'=-0.03 and energy>=0.0;
  transitions
    charged -[then voltage:=energy/50.0+4.0]-> charged;
    charged -[empty when energy<=20.0]-> depleted;
    depleted -[then voltage:=energy/50.0+4.0]-> depleted;
end Battery.Imp;
```


Modeling the redundant power system

Power system with battery subcomponents: Addin reconfiguration: Adding port connections:




```
system Power
  features
    voltage: out data port real;
end Power;

system implementation Power.Imp
  subcomponents
    batt1: device Battery.Imp in modes (primary);
    batt2: device Battery.Imp in modes (backup);
  connections
    data port batt1.voltage -> voltage in modes (primary);
    data port batt2.voltage -> voltage in modes (backup);
  modes
    primary: initial mode;
    backup: mode;
  transitions
    primary -[batt1.empty]-> backup;
    backup -[batt2.empty]-> primary;
end Power.Imp;
```


Deviations from AADL

Omissions

Some advanced features of AADL such as property associations, component refinement, prototypes, `event data` ports, `in out` ports, ...

Simplifications

(multi-way) synchronous communication (rather than asynchronous channel communication).

Extensions

- default values for data elements
- support for mode/error state history (upon component re-activation)
- hybridity, i.e., mode invariants, trajectory equations
- specification of observability requirements

Event-data automata

Definition (Event-data automaton)

An event-data automaton (EDA) is a tuple

$$\mathfrak{A} = (M, X, V, \iota, E, \rightarrow)$$

with

- M finite set of modes ({`charged`, `depleted`})
- $X = IX \uplus OX \uplus LX$ finite set of input/output/local variables (`voltage`, `energy`)
- $V := \{v \mid v : X \to \ldots\}$ valuations
- $\iota : M \to (V \to \mathbb{B})$ mode invariants (`while energy>=20.0`)
- $E = IE \uplus OE$ finite set of input/output events (`empty`)
- $\rightarrow \subseteq M \times \underbrace{(E \cup \{\tau\})}_{\text{trigger}} \times \underbrace{(V \to \mathbb{B})}_{\text{guard}} \times \underbrace{(V \to V)}_{\text{effect}} \times M$ transition relation (`when energy<=20.0`, `then voltage:=...`)

Networks of event-data automata

Dynamic reconfiguration

$\Longrightarrow$ component activity and port connections mode dependent

Definition (Networks of Event-Data Automata)

A network of event-data automata (NEDA) is a tuple

$$\mathfrak{N} = ((\mathfrak{A}_i)_{i \in [n]}, \alpha, EC, DC)$$

with $n \geqslant 1$, $[n] := \{1, \ldots, n\}$, and

- each $\mathfrak{A}_i$ an EDA $\mathfrak{A}_i = (M_i, m_0^i, X_i, v_0^i, \iota_i, E_i, \rightarrow_i)$
- $M := \prod_{i=1}^{n} M_i$ set of global modes
- $\alpha : M \rightarrow 2^{[n]}$ activation mapping
- $EC : M \rightarrow (\{i.e \mid i \in [n], e \in E_i\})^2$ event connection mapping
- $DC : M \rightarrow (\{i.x \mid i \in [n], x \in X_i\})^2$ data connection mapping

Operational semantics of networks of EDAs

Example (Power system)

⟨m = primary, v = 6.0⟩ | ⟨m = charged, e = 100.0, v = 6.0⟩ | ⟨m = charged, e = 100.0, v = 6.0⟩
⇓ 40.0
⟨m = primary, v = 6.0⟩ | ⟨m = charged, e = 20.0, v = 6.0⟩ | ⟨m = charged, e = 100.0, v = 6.0⟩
⇓ τ⟨voltage:=...⟩
⟨m = primary, v = 4.4⟩ | ⟨m = charged, e = 20.0, v = 4.4⟩ | ⟨m = charged, e = 100.0, v = 6.0⟩
⇓ τ⟨empty⟩
⟨m = backup, v = 6.0⟩ | ⟨m = depleted, e = 20.0, v = 4.4⟩ | ⟨m = charged, e = 100.0, v = 6.0⟩
⇓ 40.0
⟨m = backup, v = 6.0⟩ | ⟨m = depleted, e = 20.0, v = 4.4⟩ | ⟨m = charged, e = 20.0, v = 6.0⟩
⇓ ...

Error modeling

```
error model BatteryFailure
  features
    ok: initial state;
    dead: error state;
    batteryDied: out error propagation;
end BatteryFailure;

error model implementation BatteryFailure.Imp
  events
    fault: error event occurrence poisson 0.01;
  transitions
    ok -[fault]-> dead;
    dead -[batteryDied]-> dead;
end BatteryFailure.Imp;
```

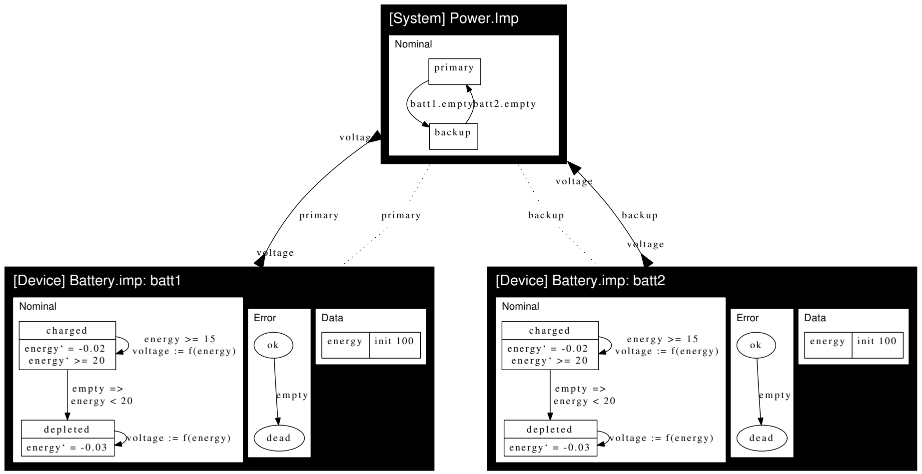
Repair

reset events (not in example) can be sent from nominal to error model of same component to attempt to repair the occurred fault.

Fault injection

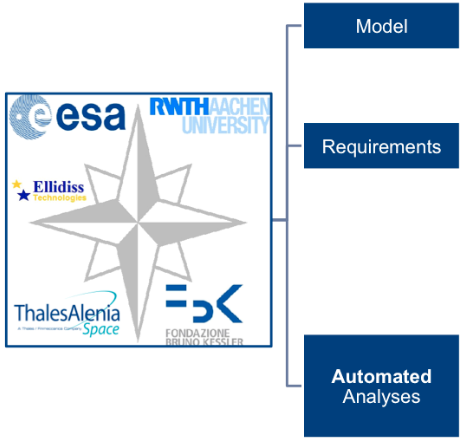
linked through fault injection

The complete power system model



Overview

Analysis: Ingredients



Property specification: Patterns, no formulas!

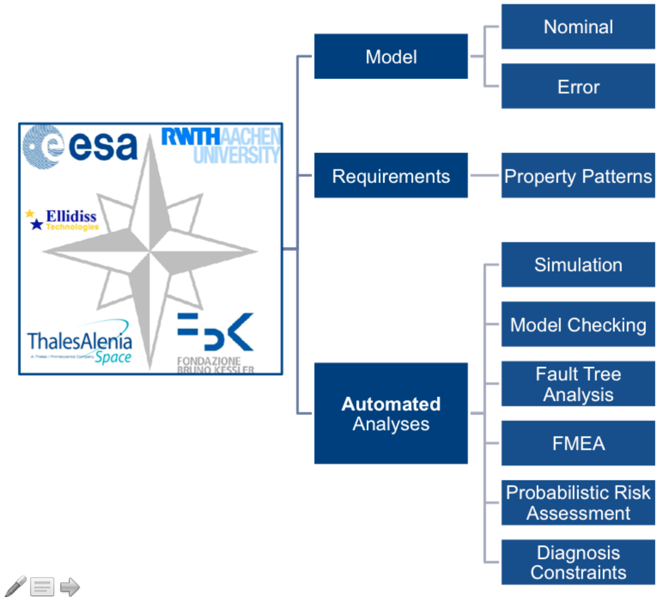
Examples

- *The system shall have a behaviour where with probability higher than p it is the case that Φ holds continously within time bound $[t_1, t_2]$.*
- *The system shall have a behaviour where Φ globally holds.*

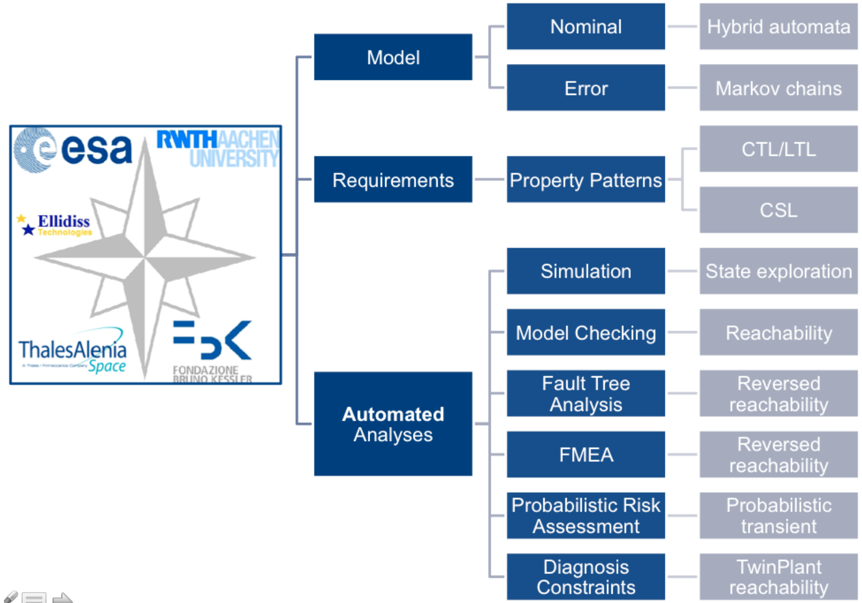
Implemented pattern systems

Formalism	Intended use	Authors
CTL, LTL	functional properties	[Dwyer et al., 1999]
MTL, TCTL	real-time properties	[Konrad & Cheng, 2005]
PCTL, CSL	probabilistic properties	[Grunske, 2008]

Analysis: Techniques



Analysis: Models, Logics, and Algorithms



Failure Mode and Effects Analysis (FMEA)

Main features

- Inductive technique (bottom-up)
- Tabled representation of fault effects on system properties
- Widespread use in aerospace, avionics, and other domains

Example FMEA table

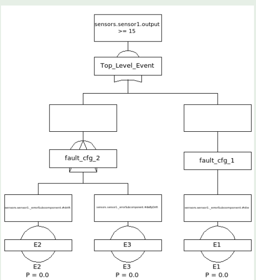
Ref. No.	*Item*	*Failure Mode*	*Failure Cause*	*Local Effects*	*System Effects*	*Detection Means*	*Severity*	*Corrective Actions*
1	Pump	Fails to operate	Comp. broken No input flow	Coolant temperature increases	Reactor temperature increases	Temperature alarm	Major	Start secondary pump Switch to secondary circuit
2	Valve	Stuck closed	Comp. broken	Excess liquid	Reactor pressure increases	Coolant level sensor	Critical	Open release valve
3		Stuck open	Comp. broken	Insufficient liquid	Reactor temperature increases	Coolant level sensor	Critical	Open tank valve

Fault Tree Analysis (FTA)

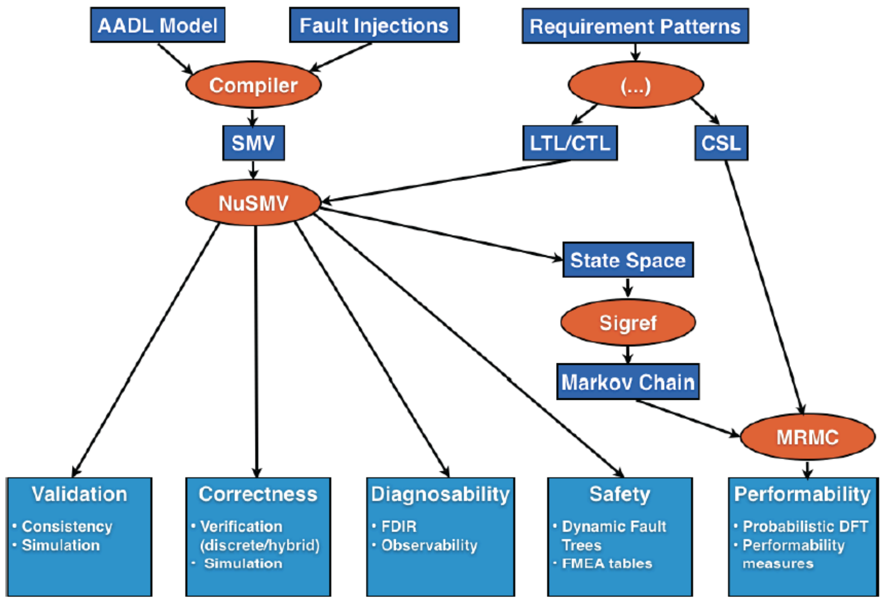
Dynamic FTs

- Dynamic FTs extend FTs by considering dynamic aspects, such as: ordering constraints, functional dependencies, spares
- Dynamic FTs in COMPASS: priority ANDs (PANDs)

Example Dynamic Fault Tree



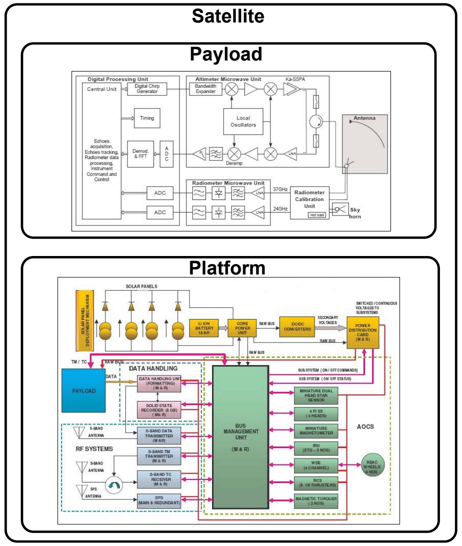
Tool architecture



Overview

Case study: Satellite of project [redacted]

Launches between 2012-2020



Payload is mission-specific equipment, e.g.:

- telecom transponders,
- navigation signals,
- earth observation telemetry (weather, radiation, salinity).

Platform keeps the satellite orbiting in space, consists of:

- attitude & orbital control
- power distribution
- data handling
- communications
- thermal regulation

AADL model of satellite platform

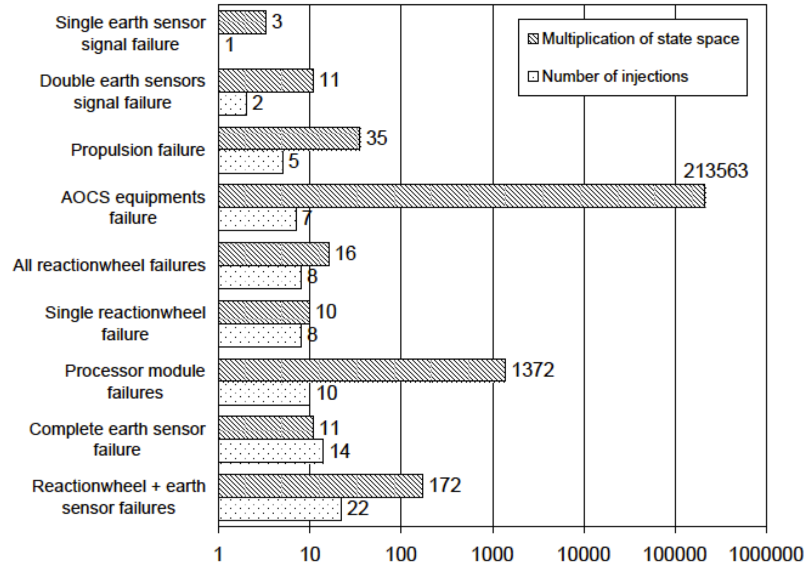
Verification & validation objectives

- Ensure nominal and degraded conditions are handled correctly by the fault management system.
- Ensure performance and risks are within specified limits.

Model characteristics

Scope	Metric	Count
Model	Components	86
	Ports	937
	Modes	244
	Error models	20
	Recoveries	16
	Nominal state space	48421100
	LOC (without comments)	3831
Requirements	Propositional	25
	Absence	2
	Universality	1
	Response	14
	Probabilistic Invariance	1
	Probabilistic Existence	1

State space growth by fault injection



Analysis results[2]

Analysis	Fault injection	Time (in s)	Memory (in MB)
LTL model checking	none	224	122
LTL model checking	single sensor failure	296	125
Hybrid BMC (depth 70)	single sensor failure	2176	1006
Fault tree analysis (TLE)	double sensor failure	555	134
Fault tree analysis (TLE)	AOCS equipment failure	2898	181

Analysis	Fault injection	Time (in s)	Memory (in MB)
Dynamic FTA	AOCS equipment failure	5581	212
FMEA table generation	double sensor failure	1003	134
Fault detection analysis	double sensor failure	1173	142
Diagnosability analysis	double sensor failure	586093*	1474
Performability analysis	double sensor failure	33166*	2103

[2]Setup: Intel Xeon 2.33 GHz machine with 16 GB RAM.

Overview

Summary

Achievements:

- Component-based model framework based on AADL
- Novelties: hybrid, error modeling, dynamic reconfigurations, ...
- Automated correctness, safety, and performability analysis
- Industrial evaluations showed maturity

In a nutshell: trustworthy aerospace design := AADL modeling + analysis

Future and current activities:

- Compositional model checking (ESA funded)
- Security and compositionality aspects in AADL (EU funded)
- Automated test generation from AADL models

Further information

- Overview paper (Yushstein et. al, IEEE SMC-IT 2011)
- AADL formal semantics (Bozzano et. al, Computer J. 2011)
- Slicing of AADL specifications (Odenbrett et. al, NASA FM 2010)
- AADL model checker (Bozzano et. al, CAV 2010)
- Our variant of the AADL language (Bozzano et. al, MEMOCODE 2009)
- ESA case study (Esteve et. al, ICSE 2012)
- Tool download at http://compass.informatik.rwth-aachen.de/