iFM & ABZ 2012 Tutorial
Safety, Dependability and Performance Analysis of Extended AADL Models

Part 1: Overview

European Space Agency
European Space Research and Technology Centre

RWTH Aachen University
Software Modeling and Verification Group
Joost-Pieter Katoen & Thomas Noll

Fondazione Bruno Kessler
Centre for Scientific and Technological Research
Marco Bozzano & Alessandro Cimatti

iFM & ABZ 2012; June 18, 2012; Pisa, Italy

Outline of Tutorial

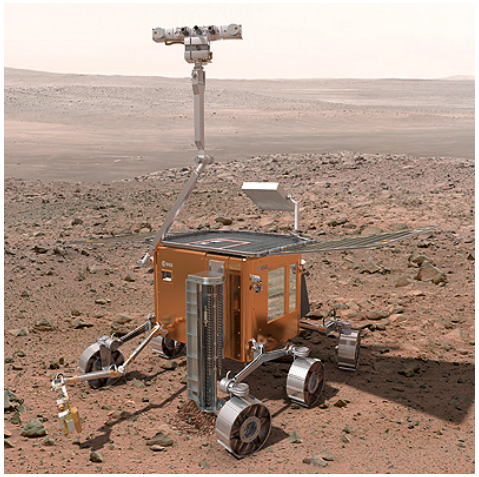
1. Overview
2. System Modeling Using AADL
3. Checking Functional Correctness

   Coffee Break
4. Safety and Dependability Analysis
5. Fault Detection, Isolation and Recovery (FDIR) Analysis
6. Performability Evaluation

Contents of Overview

Outline

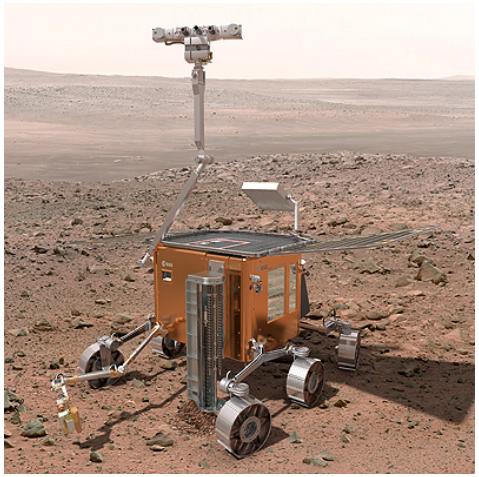
Domain: Fault-Tolerant Space System Architectures



ExoMars Rover: autonomy

- 4 to 21 min. for radio latency to earth
- [redacted] Martian days autonomous survival

Domain: Fault-Tolerant Space System Architectures



ExoMars Rover: autonomy

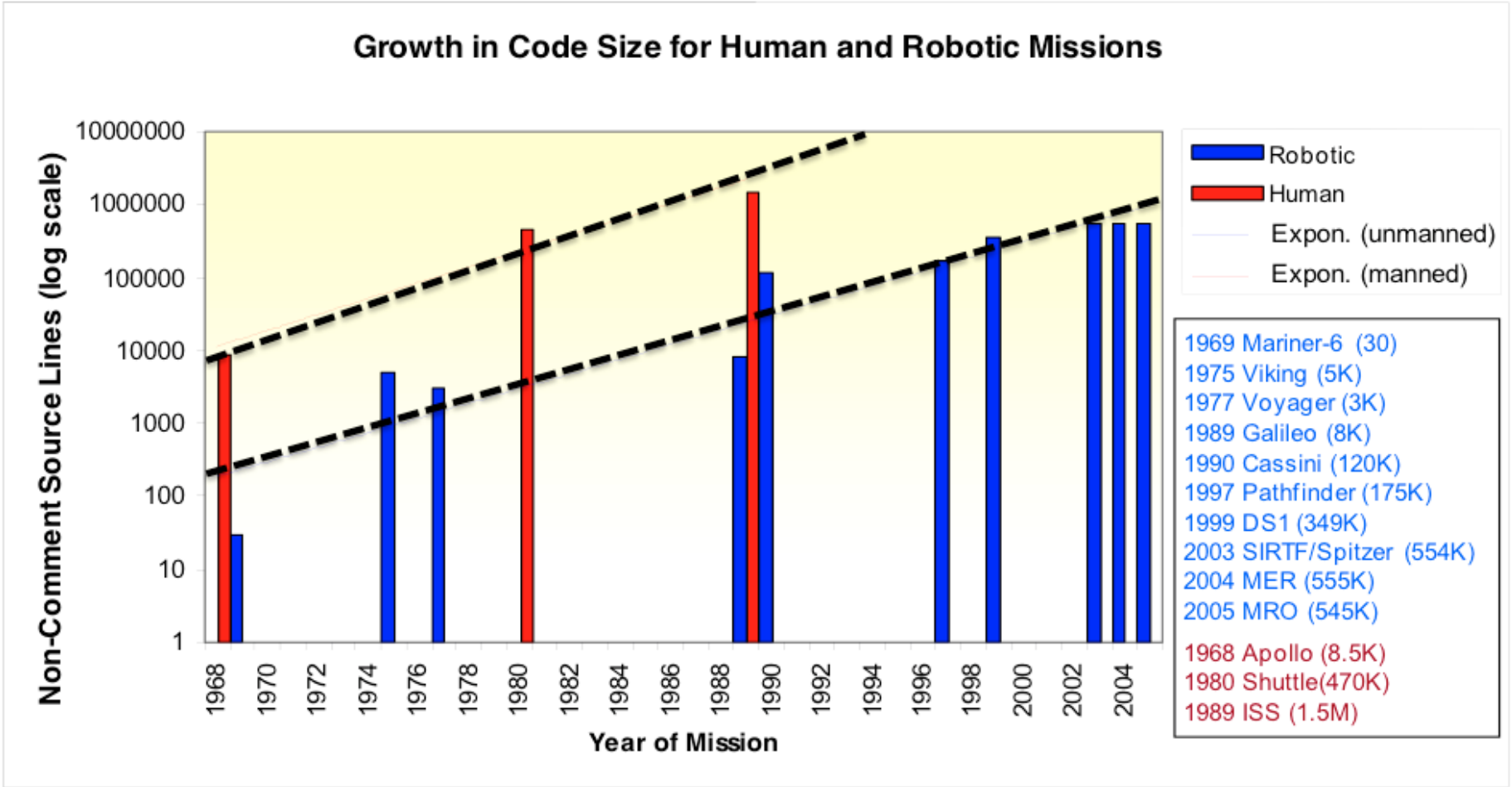
- 4 to 21 min. for radio latency to earth
- [redacted] Martian days autonomous survival

Autonomous Transfer Vehicle (ATV):
autonomy and safety

- fully-automated navigation and docking to ISS
- human-rated requirements for safety (of ISS)

$\Rightarrow$ multi-failure tolerance (1 MLOC of control code)

Spacecraft := Flying Software

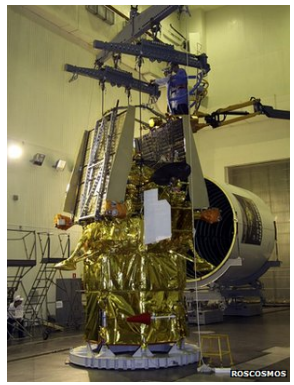


NASA Study Flight Software Complexity (2009)

Extreme Dependability!

Requirements

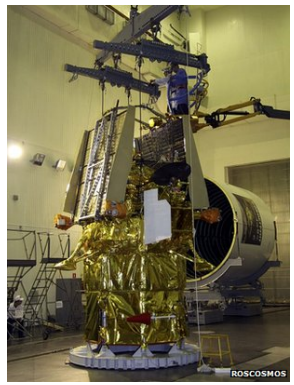
- Must offer service without interruption for a very long time – typically years or decades
- Faults are costly and may severely damage reputations:
  - Ariane 5 crash in 1996 due to arithmetic overflow
  - Launch failure of recent Phobos-Grunt sample return mission
- "Five nines" (99.999 %) dependability not sufficient



Extreme Dependability!

Requirements

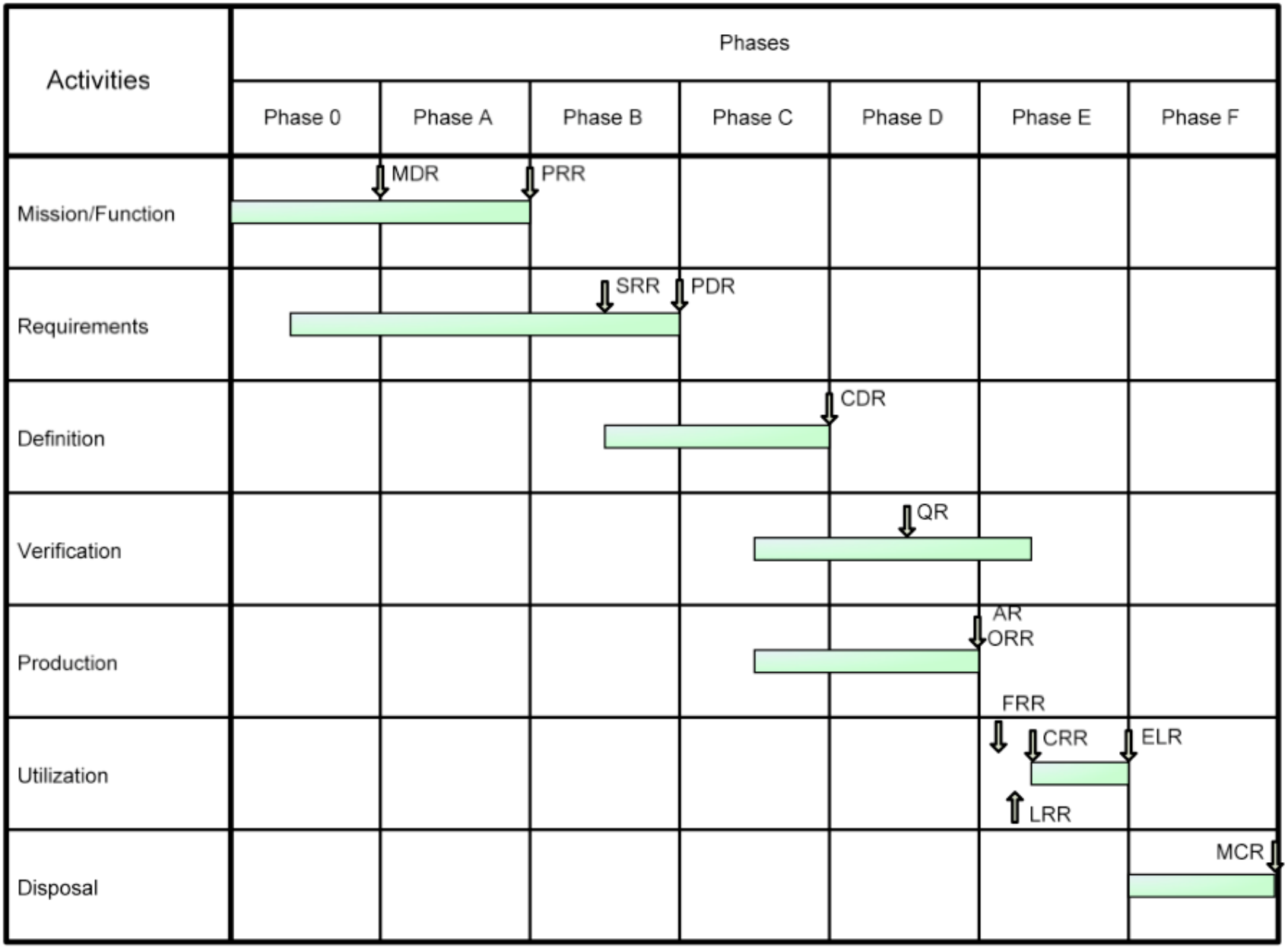
- Must offer service without interruption for a very long time – typically years or decades
- Faults are costly and may severely damage reputations:
  - Ariane 5 crash in 1996 due to arithmetic overflow
  - Launch failure of recent Phobos-Grunt sample return mission
- "Five nines" (99.999 %) dependability not sufficient



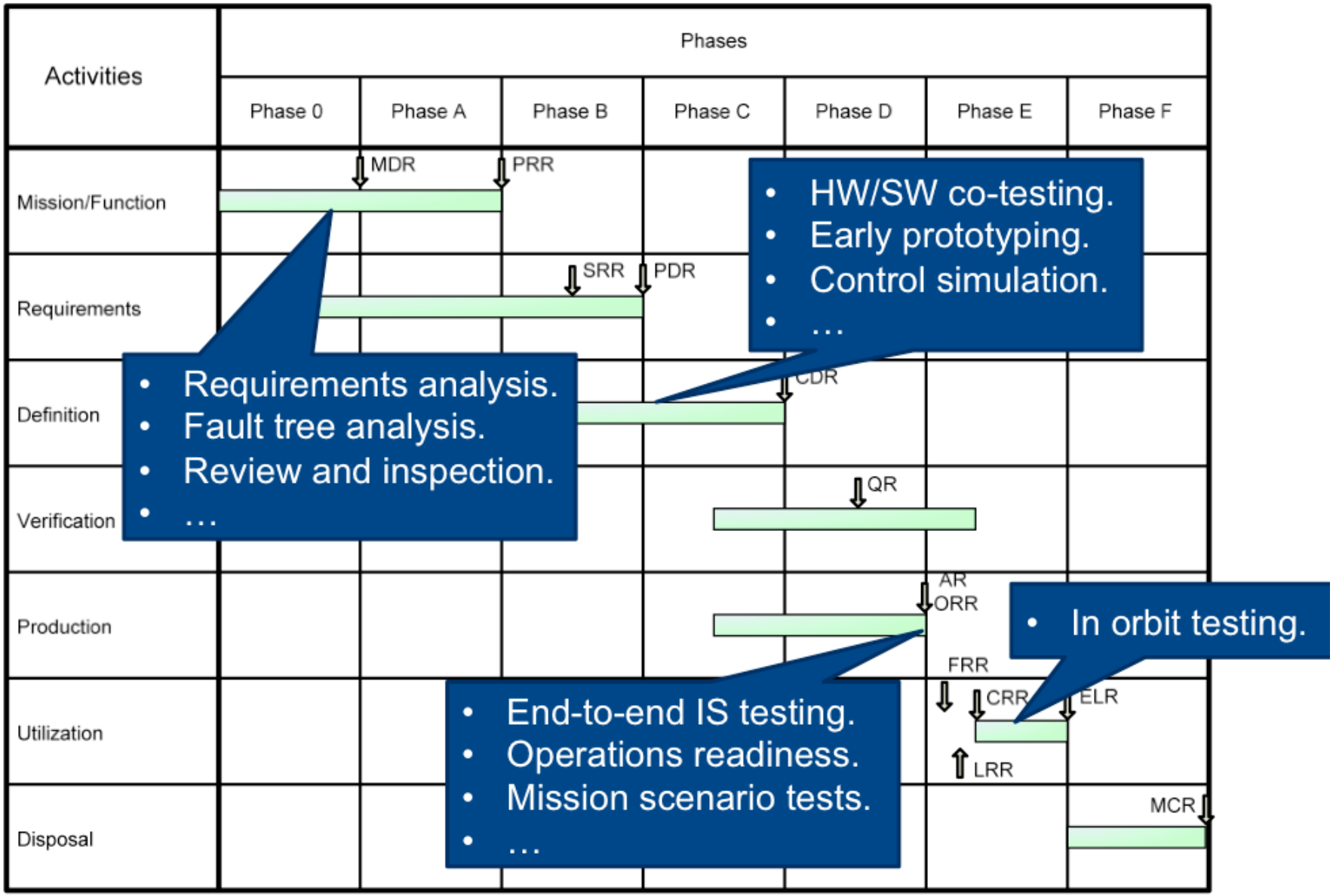
Challenges

- 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
- Effect of Fault Diagnosis, Isolation and Recovery (FDIR) measures must be quantifiable

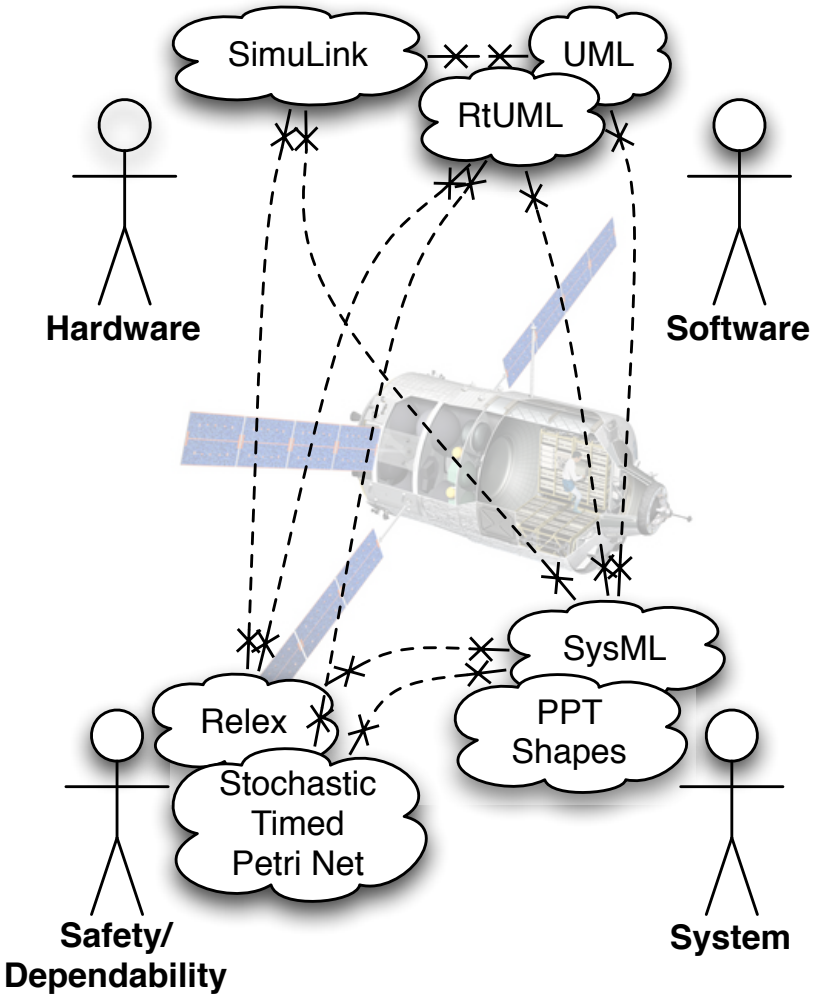
Spacecraft Design Process



Verification and Validation in Spacecraft Design



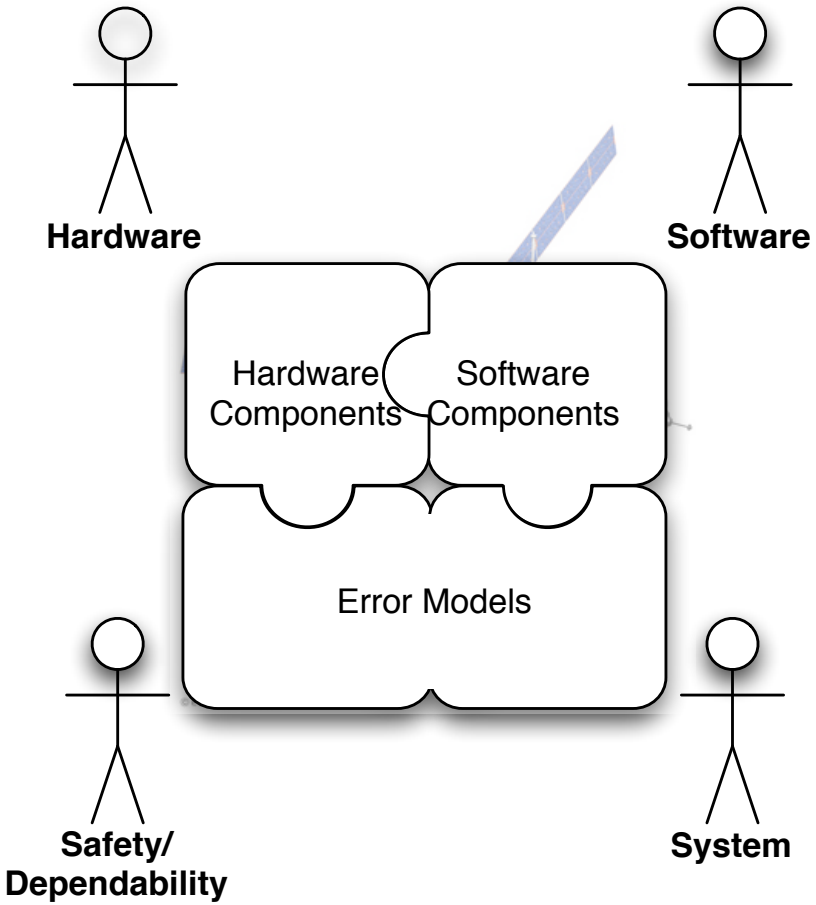
Current Limitations



Limitations

- HW verified independently of SW with exaggerated mutual assumptions
- Safety & dependability analyses isolated from HW/SW models
- Multiple modeling formalisms for different system aspects (e.g. real-time, probabilistic, hybrid)
- No coherent approach to study effectiveness of FDIR

Possible Solutions



Solutions

Combination of

- HW, SW and their bindings +
- real-time, hybrid and probabilistic aspects +
- error models +
- non-nominal modes

in a single integrated model

Outline

The COMPASS mission

Develop a model-based approach to system-software co-engineering while focusing on a coherent set of modeling and analysis techniques for evaluating system-level correctness, safety, dependability, and performance of on-board computer-based aerospace systems.

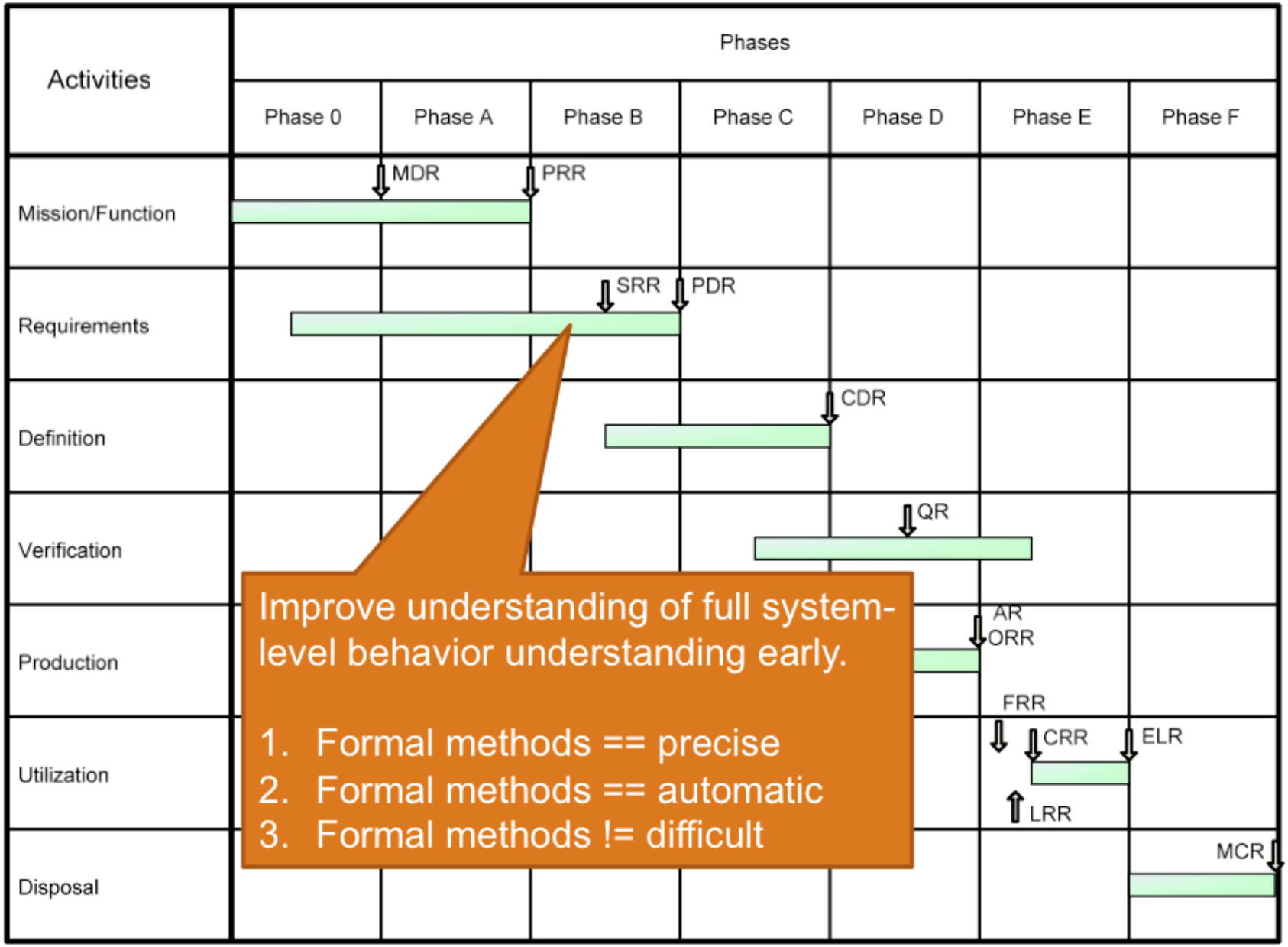
The COMPASS mission

Develop a model-based approach to system-software co-engineering while focusing on a coherent set of modeling and analysis techniques for evaluating system-level correctness, safety, dependability, and performance of on-board computer-based aerospace systems.

Derived objectives

1. Modeling formalism: variant of AADL (SAE Architecture Analysis and Design Language)
2. Verification methodology based on state-of-the-art formal methods
3. Toolset supporting the analysis of AADL models
4. Evaluation on industrial-size case studies from aerospace domain

Increase Formality in Spacecraft Design



COMPASS Project Partners

Consortium

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

Funding & supervision

- European Space Agency

COMPASS Project Phases

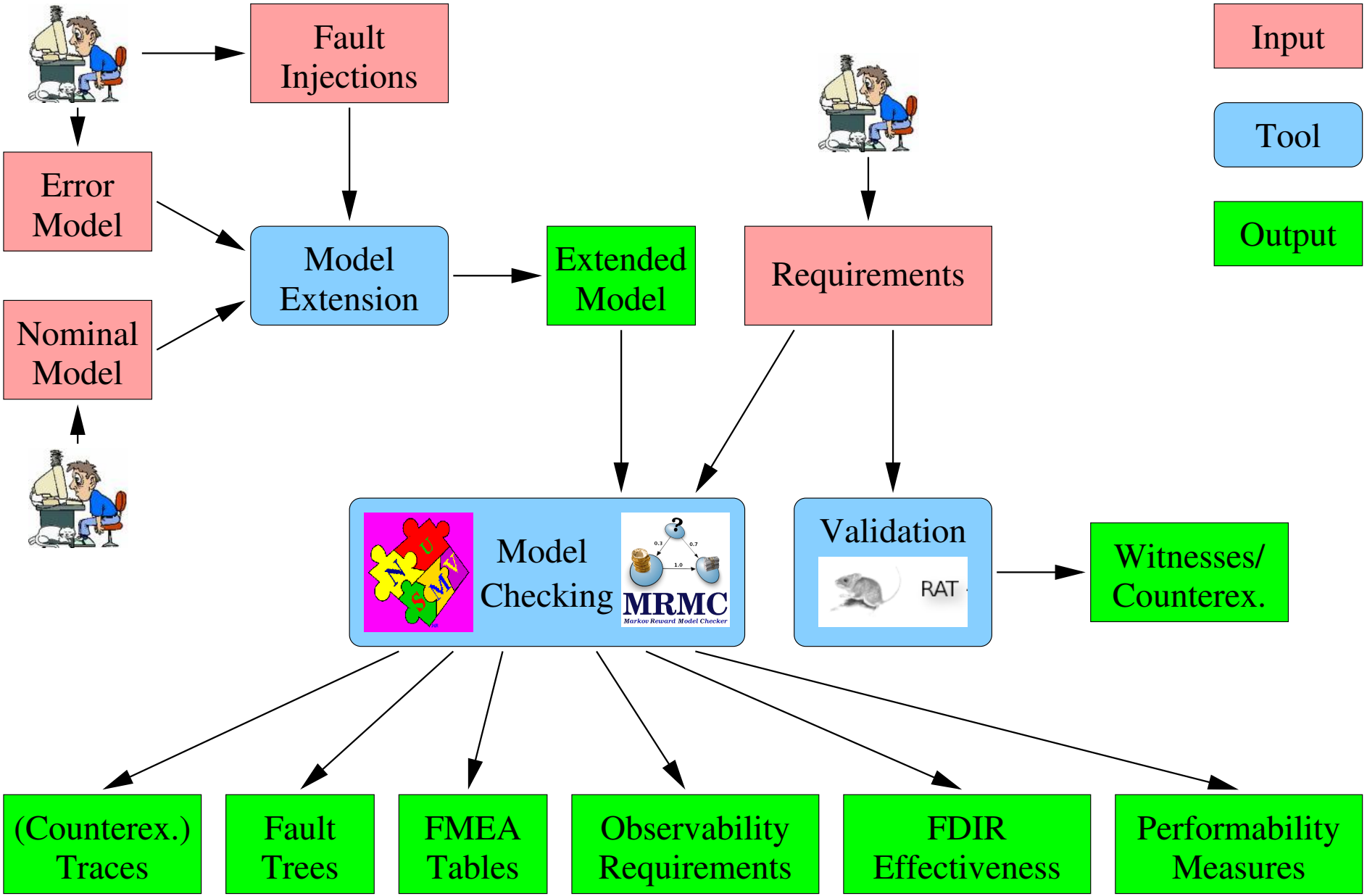
1. Project kick-off — February 2008
2. Language design
3. Software tool specification + software design document
4. Formal semantics — October 2008
5. Prototype tool implementation — April 2009
6. Prototype evaluation
7. Final tool implementation — December 2009
8. Final tool evaluation — March 2010
9. Project extension — until March 2011
10. New projects (NPI, CGM) — until December 2011

COMPASS Project Phases

1. Project kick-off — February 2008
2. Language design
3. Software tool specification + software design document
4. Formal semantics — October 2008
5. Prototype tool implementation — April 2009
6. Prototype evaluation
7. Final tool implementation — December 2009
8. Final tool evaluation — March 2010
9. Project extension — until March 2011
10. New projects (NPI, CGM) — until December 2011

Total budget: $\approx$ 900 kEuro; $\approx$ 10 programmers involved at peak times

COMPASS Methodology



Outline

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

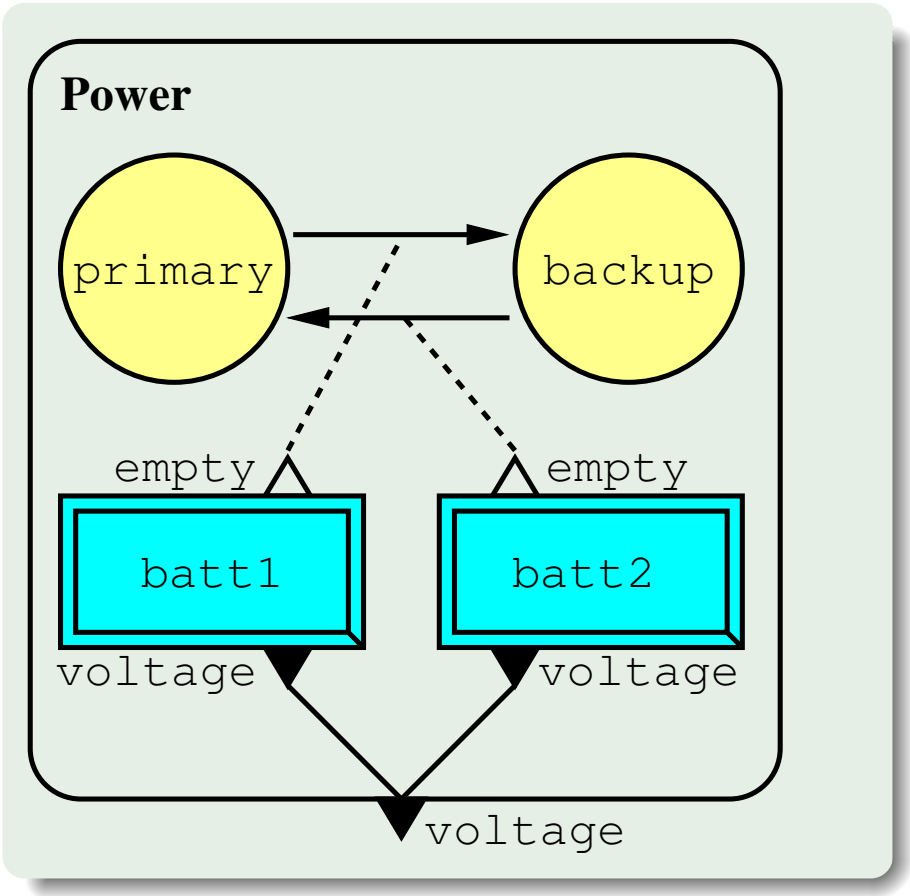
Paradigm

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

AADL Example: Redundant Power System

Redundant power system:

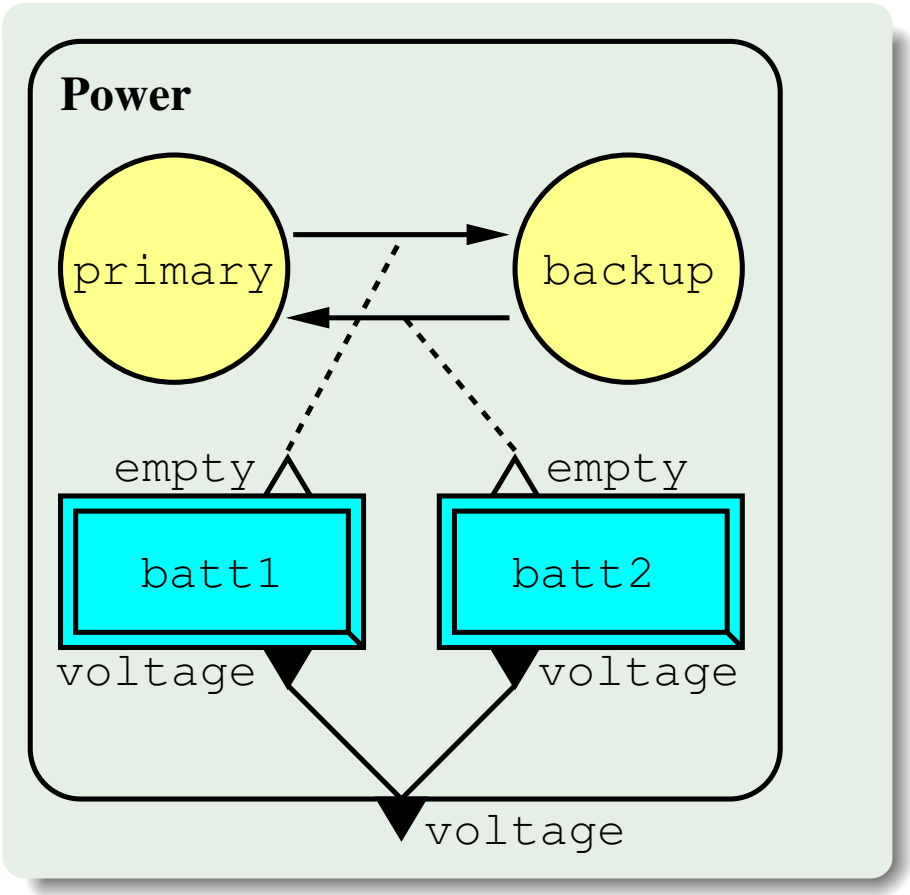
- contains two batteries `batt1`/`batt2`
- used in `primary`/`backup` mode
- power switches from `primary` to `backup` (and back) when `batt1` (`batt2`) empty
- additionally provides `voltage` information



AADL Example: Redundant Power System

Redundant power system:

- contains two batteries `batt1`/`batt2`
- used in `primary`/`backup` mode
- power switches from `primary` to `backup` (and back) when `batt1` (`batt2`) empty
- additionally provides `voltage` information



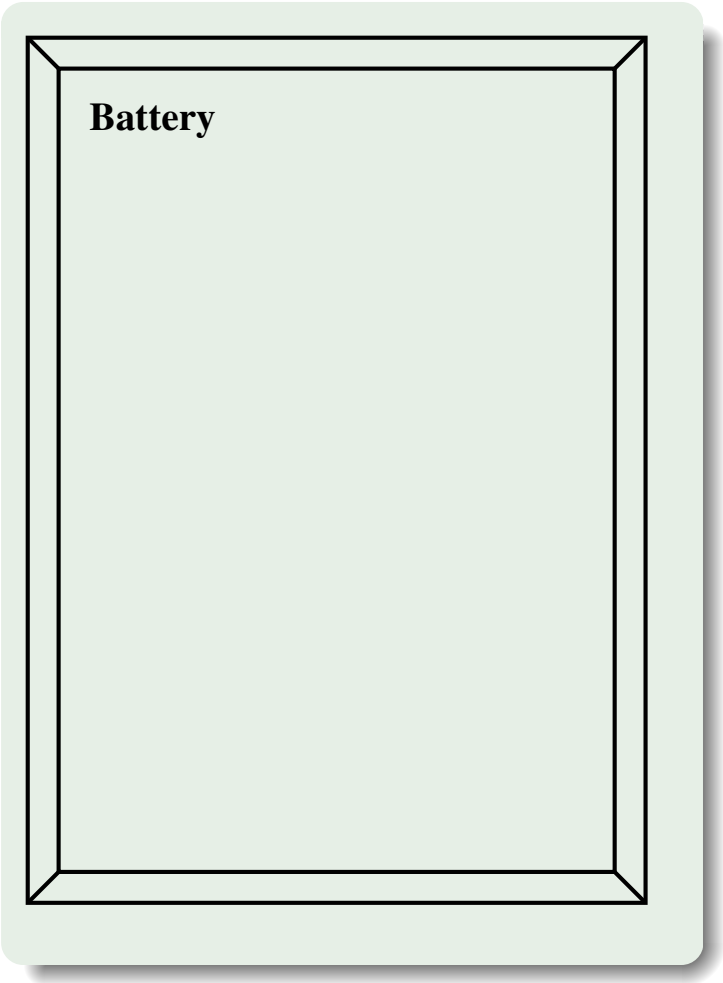
We shall show:

- hybrid behavior of the batteries
- composition of the power system
- formalization by automata
- semantics as transition systems
- interweaving of errors

Modeling a Battery

Component type and implementation:

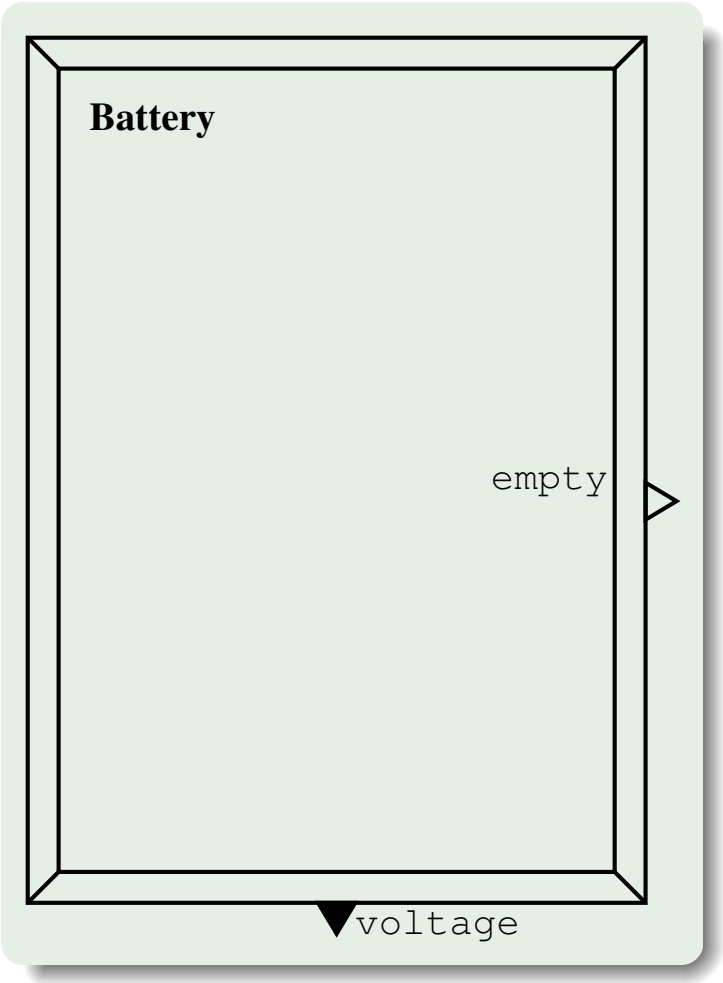
```
device type Battery


end Battery;

device implementation Battery.Imp




end Battery.Imp;
```



Modeling a Battery

Type defines the interface:

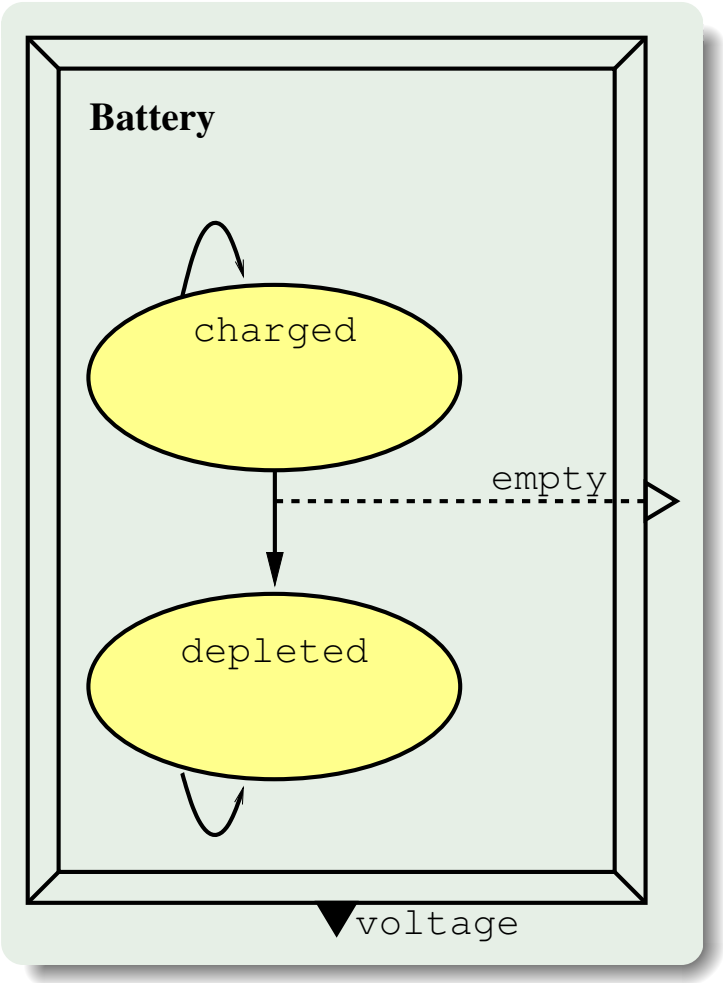
```
device type Battery
  features
    empty: out event port;
    voltage: out data port real default 6.0;
end Battery;

device implementation Battery.Imp




end Battery.Imp;
```



Modeling a Battery

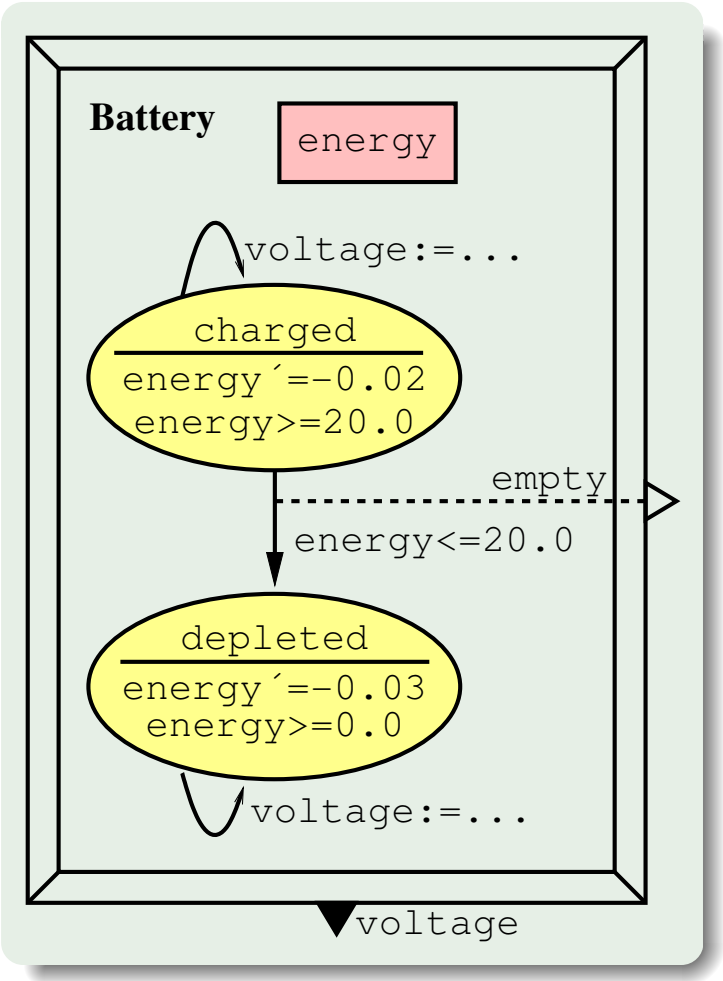
Adding modes behavior:

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

device implementation Battery.Imp


  modes
    charged: activation mode

    depleted: mode

  transitions
    charged -[]-> charged;
    charged -[empty]-> depleted;
    depleted -[]-> depleted;
end Battery.Imp;
```



Modeling a Battery

Adding hybrid behavior:

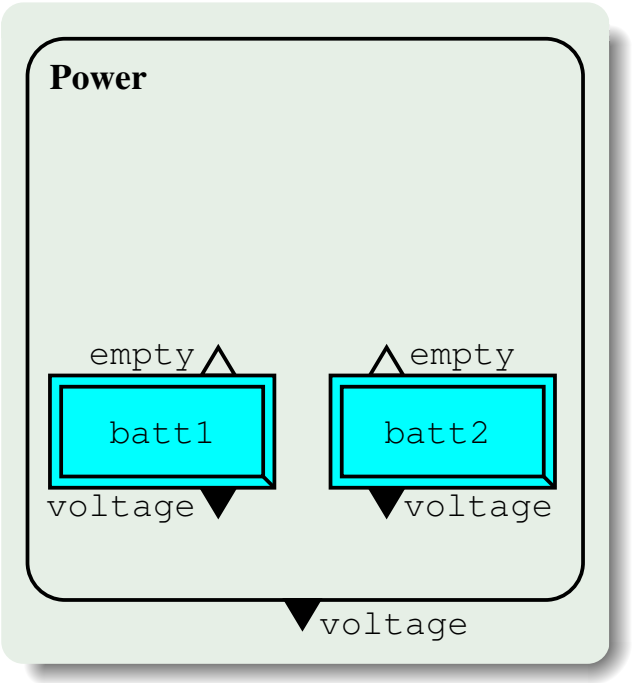



```
device type Battery
  features
    empty: out event port;
    voltage: out data port real default 6.0;
end Battery;

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:

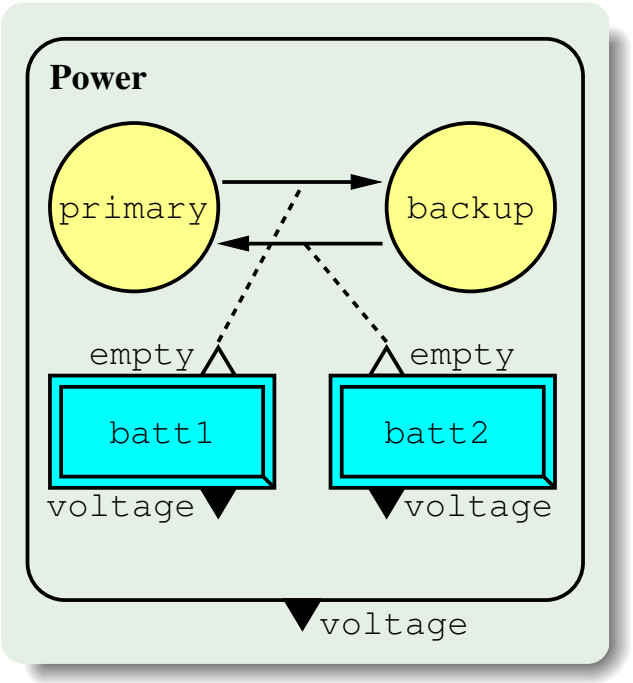
```
system Power
  features
    voltage: out data port real;
end Power;

system implementation Power.Imp
  subcomponents
    batt1: device Battery.Imp
    batt2: device Battery.Imp




end Power.Imp;
```



Modeling the Redundant Power System

Adding dynamic reconfiguration:

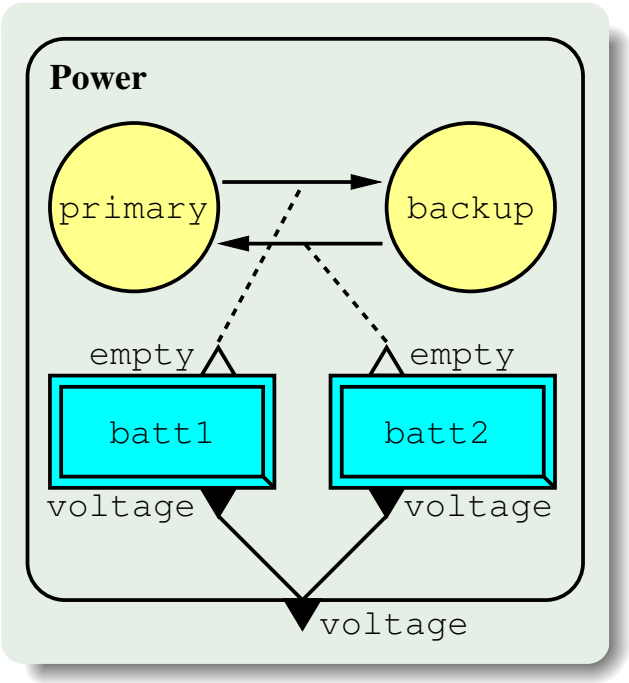
```
system Power
  features
    voltage: out data port real;
end Power;

system implementation Power.Imp
  subcomponents
    batt1: device Battery.Imp in modes (primary);
    batt2: device Battery.Imp in modes (backup);

  modes
    primary: initial mode;
    backup: mode;
  transitions
    primary -[batt1.empty]-> backup;
    backup -[batt2.empty]-> primary;
end Power.Imp;
```



Modeling the Redundant Power System

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, ...

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)

Deviations from AADL

Omissions

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

Simplifications

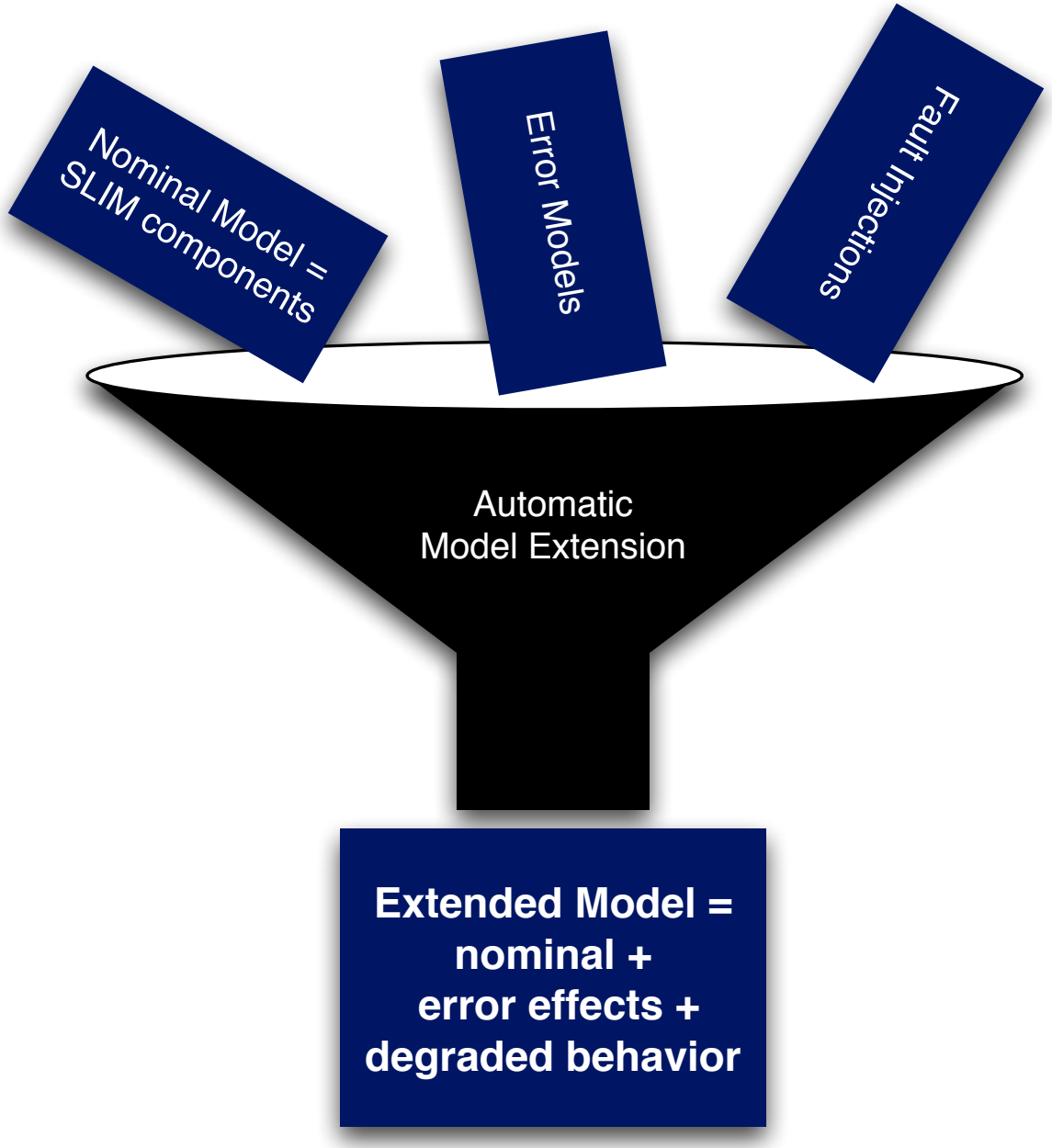
(multi-way) synchronous communication (rather than asynchronous channel communication)

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

Outline

Integrating Erroneous and Nominal Behavior



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;
```

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.

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;
```

Fault injection

An error model does not influence the nominal behavior unless they are linked through fault injection: (s, d, a) means that on entering error state s, the assignment $d := a$ is performed, where d is a data element and a the fault effect.

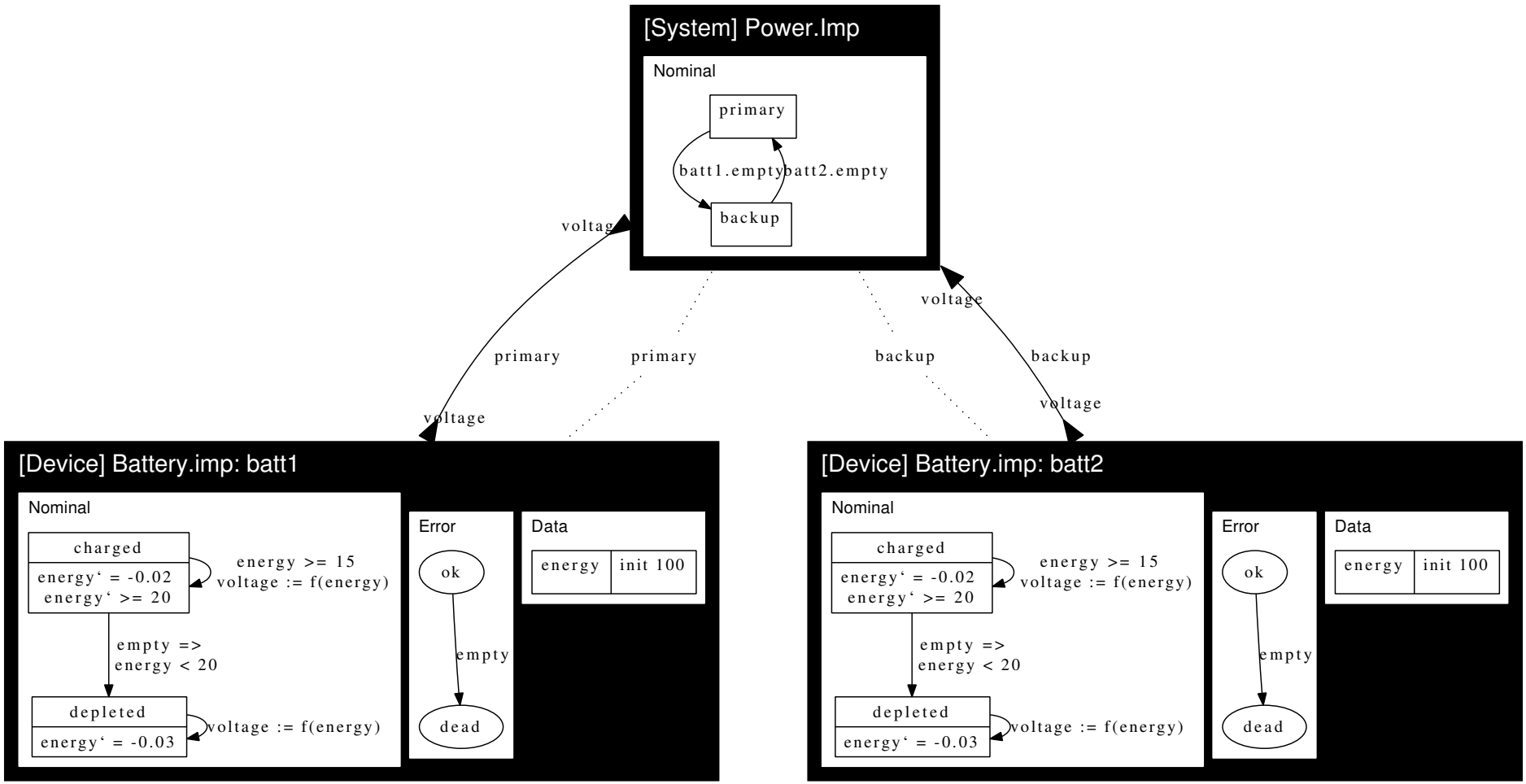
Error Modeling

```
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;
```

Fault injection

In error state `dead`, `voltage:=0`

The Complete Power System Model



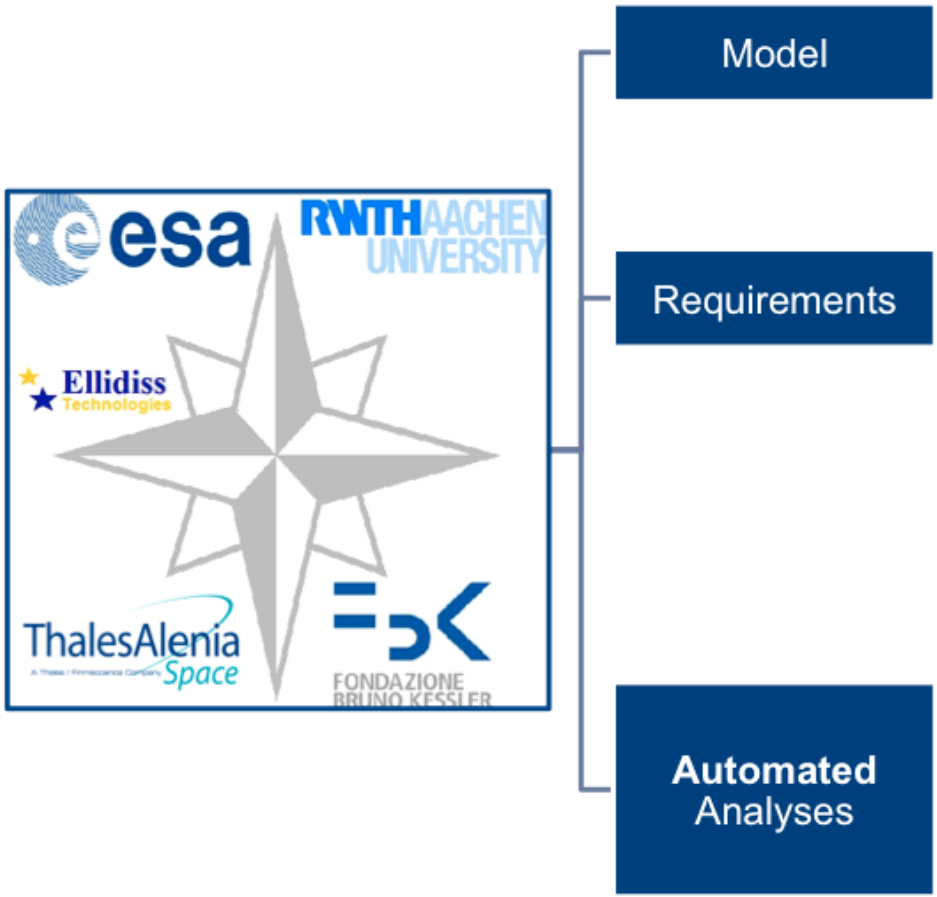
Specifying Observability

- Specification of observables for diagnosability analysis
  - for outgoing data ports of type **bool**
- Example:

```
system PowerSystem
  features
    voltage: out data port real;
    alarm: out data port bool initially false observable;
end PowerSystem;

system implementation PowerSystem.Imp
  subcomponents
    pow: system Power.Imp;
  connections
    data port pow.voltage -> voltage;
  modes
    normal: initial mode;
    critical: mode;
  transitions
    normal -[when voltage<4.5 then alarm:=true]-> critical;
    critical -[when voltage>5.5 then alarm:=false]-> normal;
end PowerSystem.Imp;
```

Outline

Analysis: Ingredients



Requirements: Patterns, not Formulas!

Patterns

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

Requirements: Patterns, not Formulas!

Patterns

- The system shall have a behavior where $80 \leq \text{voltage} \leq 90$ globally holds.
- The system shall have a behavior where with probability higher than 0.98 it is the case that $\text{voltage} \geq 80$ holds continously within time bound $[0, 10]$.

Requirements: Patterns, not Formulas!

Patterns

- The system shall have a behavior where $80 \leq \text{voltage} \leq 90$ globally holds.
- The system shall have a behavior where with probability higher than 0.98 it is the case that $\text{voltage} \geq 80$ holds continously within time bound $[0, 10]$.

(by automatic transformation)

Logic

- $\Box(80 \leq \text{voltage} \leq 90)$ (Linear Temporal Logic)
- $\mathcal{P}_{>0.98}\left(\Box^{[0,10]}(\text{voltage} \geq 80)\right)$ (Continuous Stochastic Logic)

Requirements: Patterns, not Formulas!

Logic

- $\Box(80 \leq \text{voltage} \leq 90)$ (Linear Temporal Logic)
- $\mathcal{P}_{>0.98}\left(\Box^{[0,10]}(\text{voltage} \geq 80)\right)$ (Continuous Stochastic Logic)

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

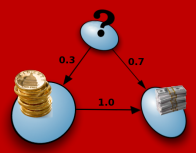
- Model
  - Nominal
  - Error
- Requirements
  - Property Patterns
- **Automated** Analyses
  - Simulation
  - Model Checking
  - Fault Tree Analysis
  - FMEA
  - Probabilistic Risk Assessment
  - Diagnosis Constraints

Analysis: Models, Logics, and Algorithms

- Model
  - Nominal → Hybrid automata
  - Error → Markov chains
- Requirements → Property Patterns
  - CTL/LTL
  - CSL
- **Automated** Analyses
  - Simulation → State exploration
  - Model Checking → Reachability
  - Fault Tree Analysis → Reversed reachability
  - FMEA → Reversed reachability
  - Probabilistic Risk Assessment → Probabilistic transient
  - Diagnosis Constraints → TwinPlant reachability

Tool Components

- Symbolic LTL and CTL model checker
- BDD- and SAT-based model checking
- SMT-based timed model checking
- Counterexample generation



MRMC

Markov Reward Model Checker

- Model checker for MRMs
- Logics: PCTL and CSL (+rewards)
- Numerical + DES engine
- Bisimulation minimisation

RAT

- Requirements analyser
- Checks logical consistency

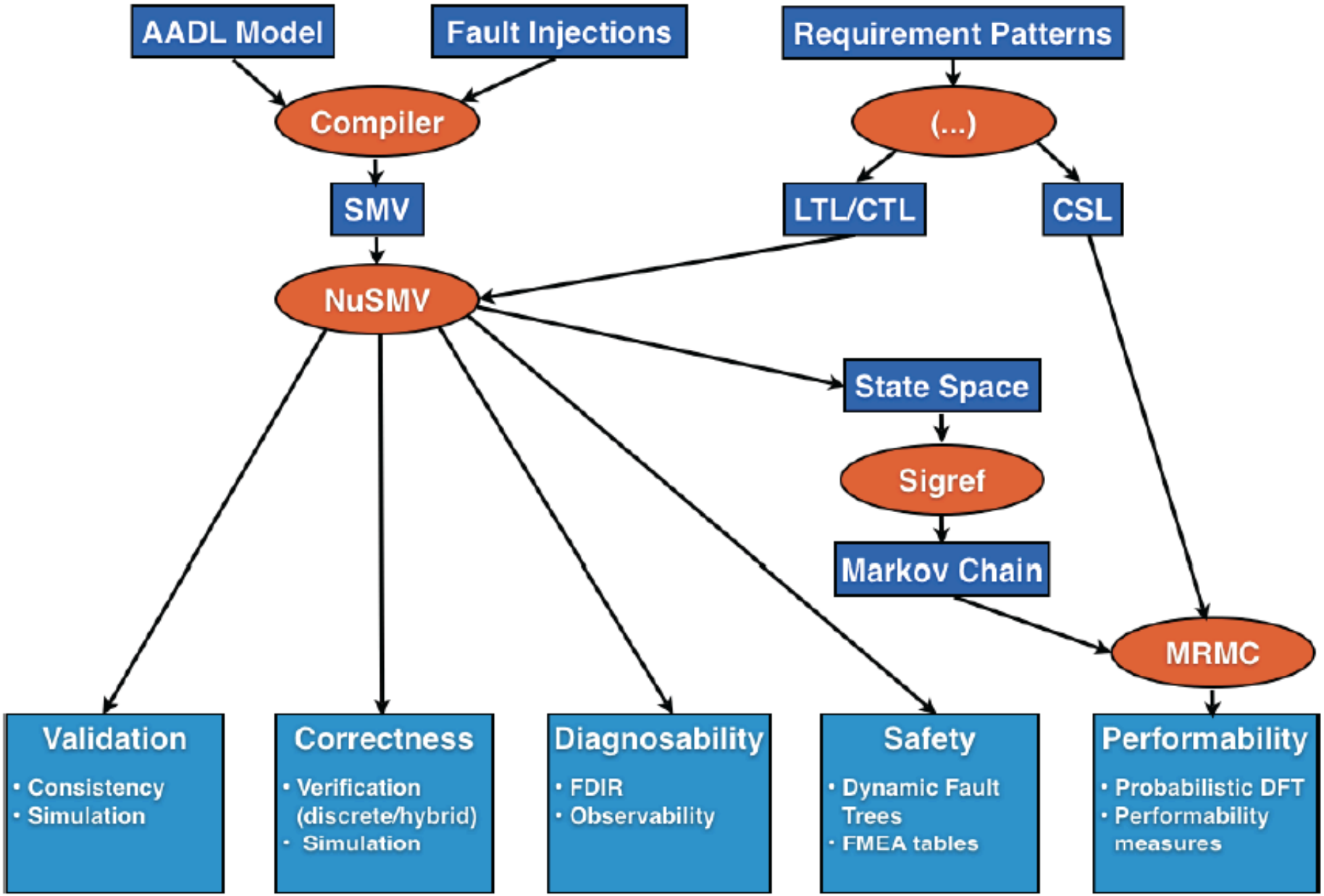
FSAP

- Safety analyser
- Fault-tree analysis

SigRef

- (MT)BDD bisimulation minimisation
- Models: Markov chains

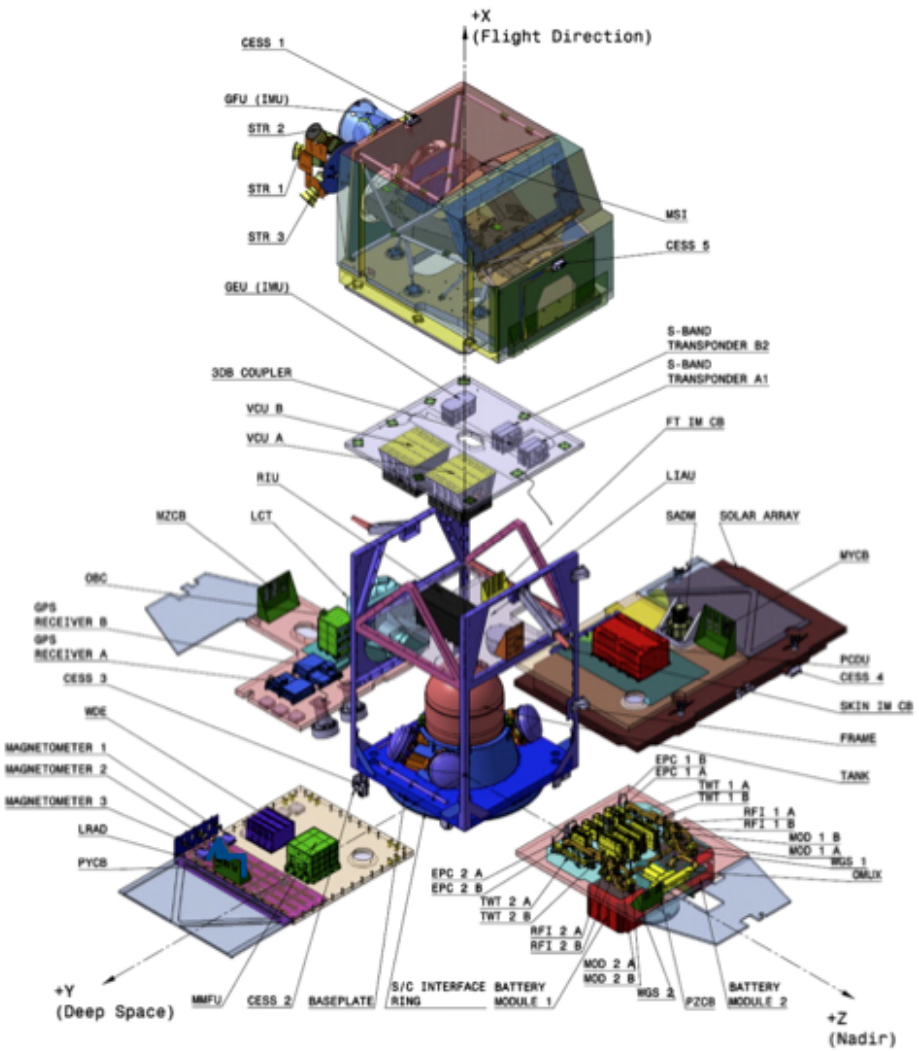
Tool Architecture



Outline

Case Study: Platform of [redacted] Satellite

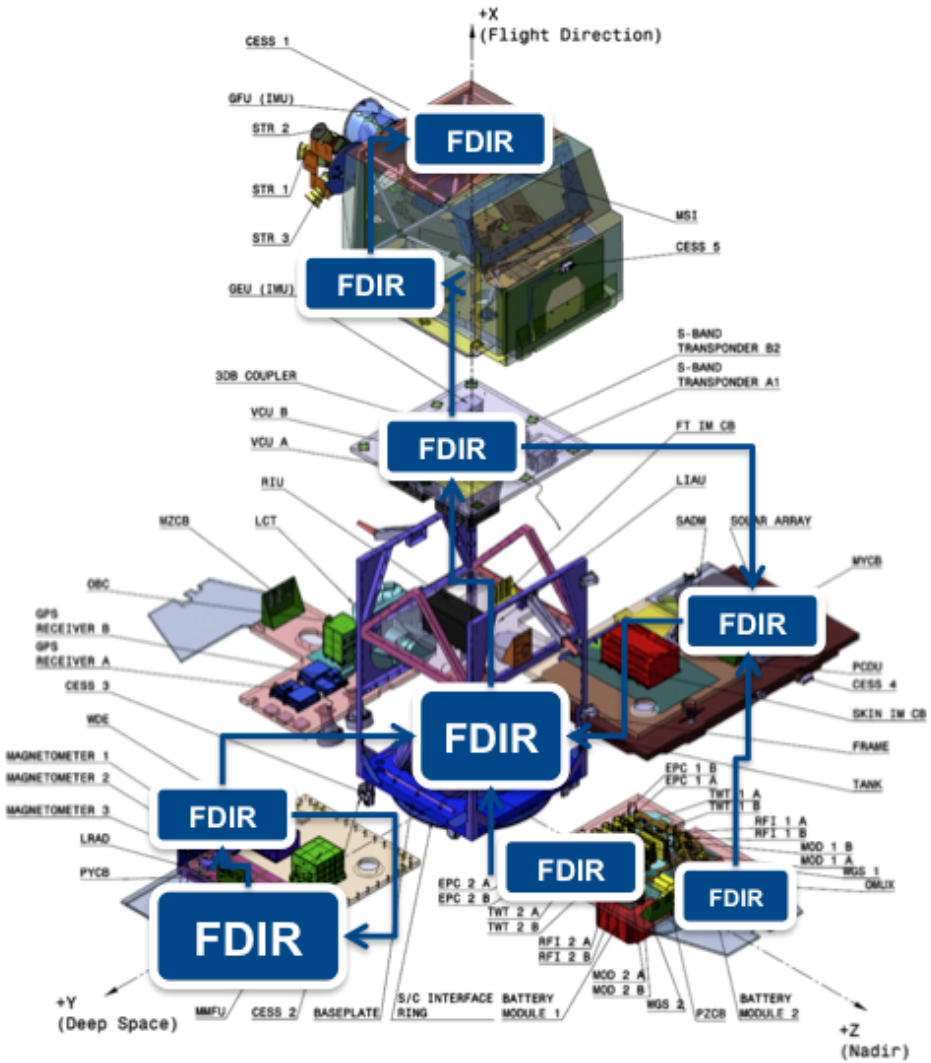
Launches between 2012-2020



- From ESA mission in development.
- Platform keeps satellite in space, like car's chassis.
  - control & data unit,
  - propulsion,
  - telemetry, tracking & cmd.
  - power,
  - attitude & orbit control sys.,
  - reconfiguration module,
  - ...

Case Study: Platform of [redacted] Satellite

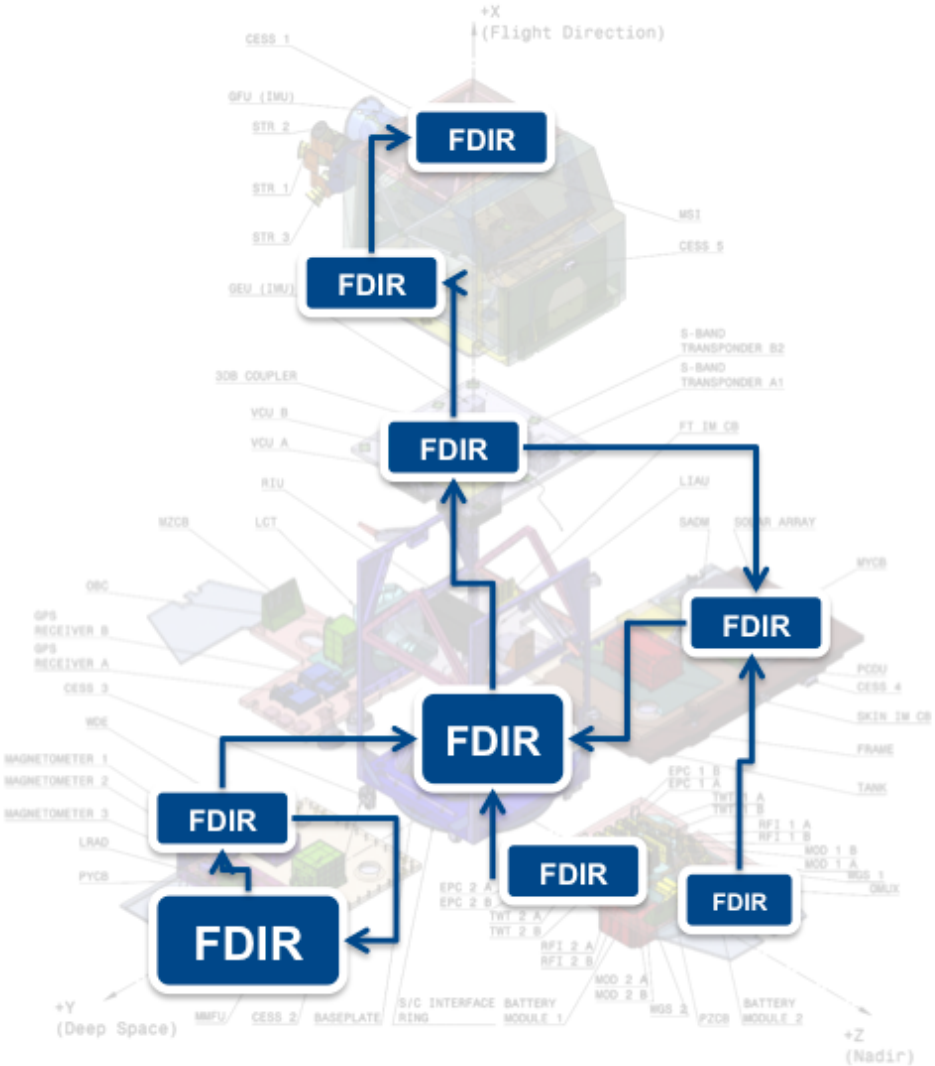
Launches between 2012-2020



- From ESA mission in development.
- Platform keeps satellite in space, like car's chassis.
  - control & data unit,
  - propulsion,
  - telemetry, tracking & cmd.
  - power,
  - attitude & orbit control sys.,
  - reconfiguration module,
  - ...

Case Study: Platform of [redacted] Satellite

Launches between 2012-2020



- From ESA mission in development.
- Platform keeps satellite in space, like car's chassis.
  - control & data unit,
  - propulsion,
  - telemetry, tracking & cmd.
  - power,
  - attitude & orbit control sys.,
  - reconfiguration module,
  - ...
- Fault detection, isolation, recovery (FDIR):
  - redundancies + recovery,
  - compensation algorithms,
  - failure isolation schemes,
  - omnipresent in satellite

AADL Model of Satellite Platform

Verification & validation objectives

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

AADL Model of Satellite Platform

Verification & validation objectives

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

Model characteristics

✓ Functional	LOC (w/o comments):	3831		
✓ Probabilistic	Components:	86	Error models:	20
✓ Real-time	Ports:	937	Recoveries:	16
✓ Hybrid	Modes:	244		

State space of nominal behavior: **48,421,100** states

AADL Model of Satellite Platform

Verification & validation objectives

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

Model characteristics

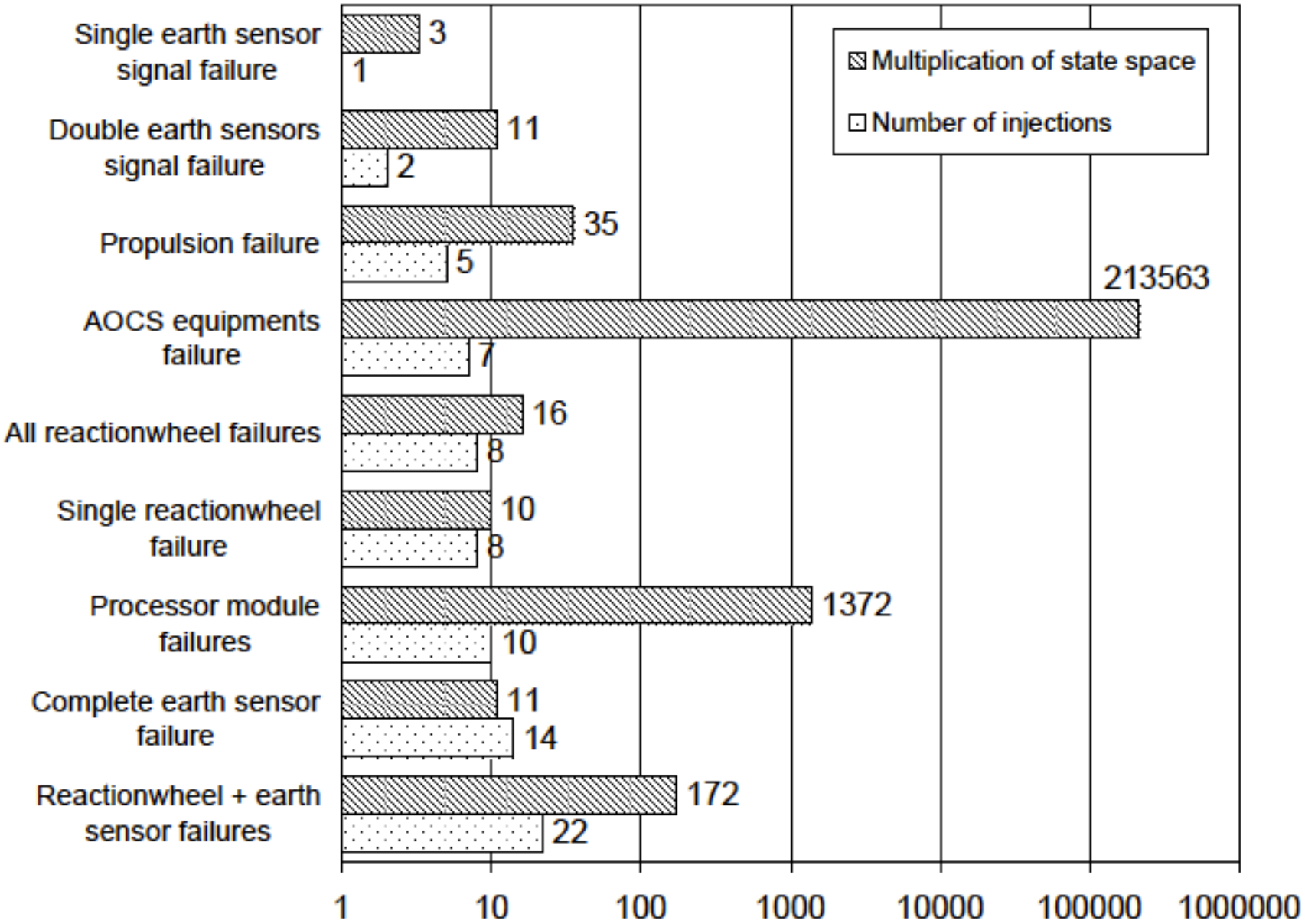
✓ Functional	LOC (w/o comments):	3831		
✓ Probabilistic	Components:	86	Error models:	20
✓ Real-time	Ports:	937	Recoveries:	16
✓ Hybrid	Modes:	244		

State space of nominal behavior: **48,421,100** states

Requirement metrics

- Functional properties: 42
  (25 propositional, 2 absence, 1 universality, 14 response)
- Probabilistic properties: 2 (1 invariance, 1 existence)

State Space Growth by Fault Injection



Analysis Results

Setup: Intel Xeon 2.33 GHz machine with 16 GB RAM

Analysis	Fault injection	Time (in sec)	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

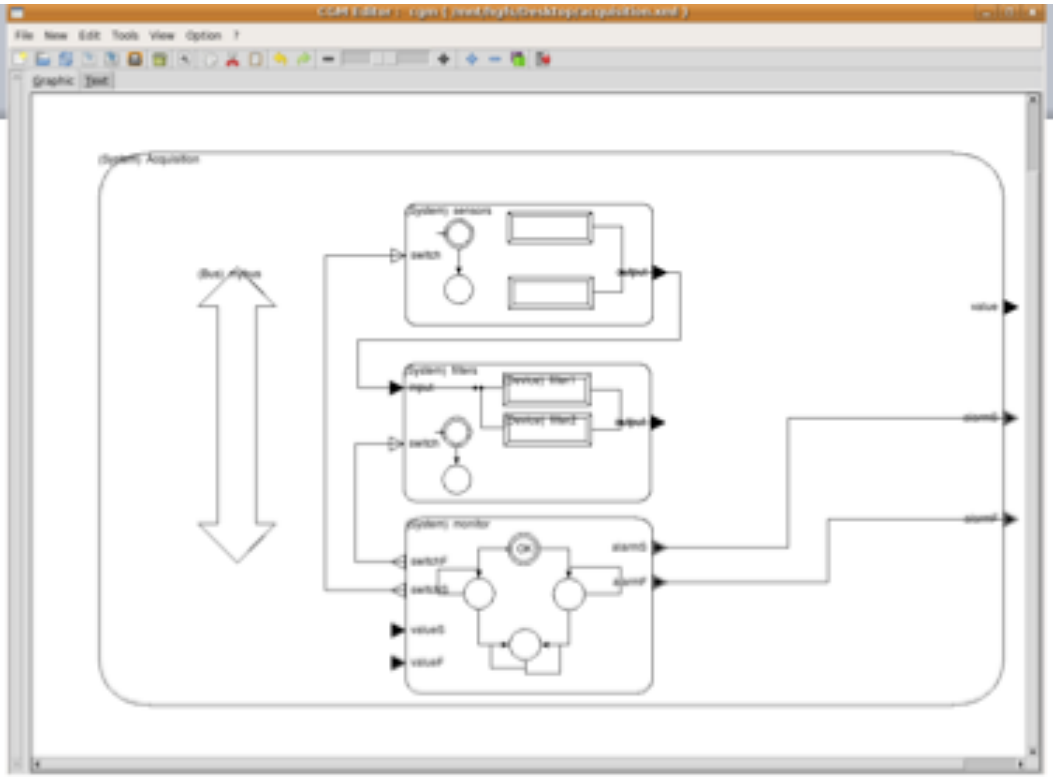
Analysis Results

Setup: Intel Xeon 2.33 GHz machine with 16 GB RAM

Analysis	Fault injection	Time (in sec)	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

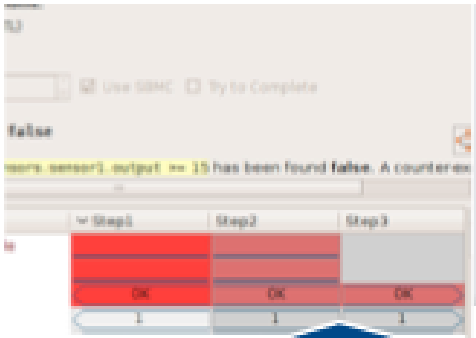
Analysis	Fault injection	Time (in sec)	Memory (in MB)
Dynamic FTA	AOCS equipment failure	5581	212
FMEA table generation	double sensor failure	1003	134
Fault detection analysis	double sensor failure	1173	142
Diagnosability analysis	double sensor failure	586093*	1474
Performability analysis	double sensor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

Lesson 1: AADL suits System Level

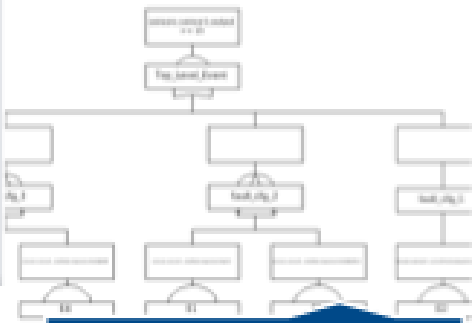


- Component-oriented and hierarchy supports refinement.
- Dynamic reconfiguration for fault tolerance.
- Integration of nominal and erroneous behavioral aspects.
- Minor suggestions for improvement reported.
- Warn for Zeno cycles, timelocks and time divergence.

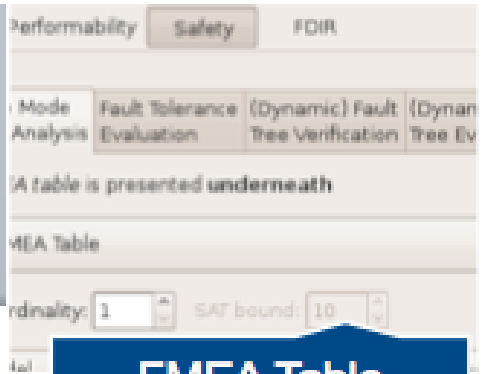
Lesson 2: Performant and Friendly Toolset



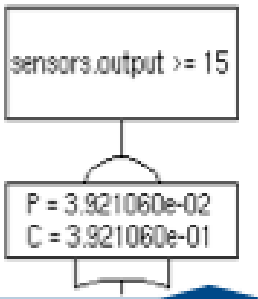
Model Checking/ Simulation



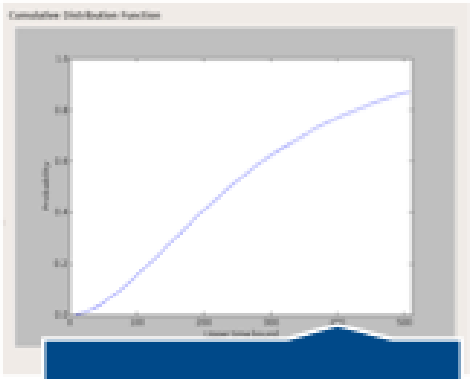
Fault Tree Generation



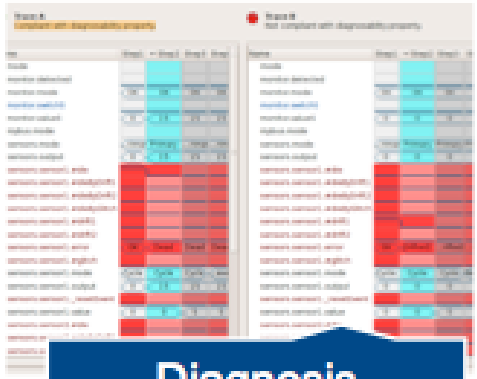
FMEA Table Generation



Probabilistic Risk Assessment



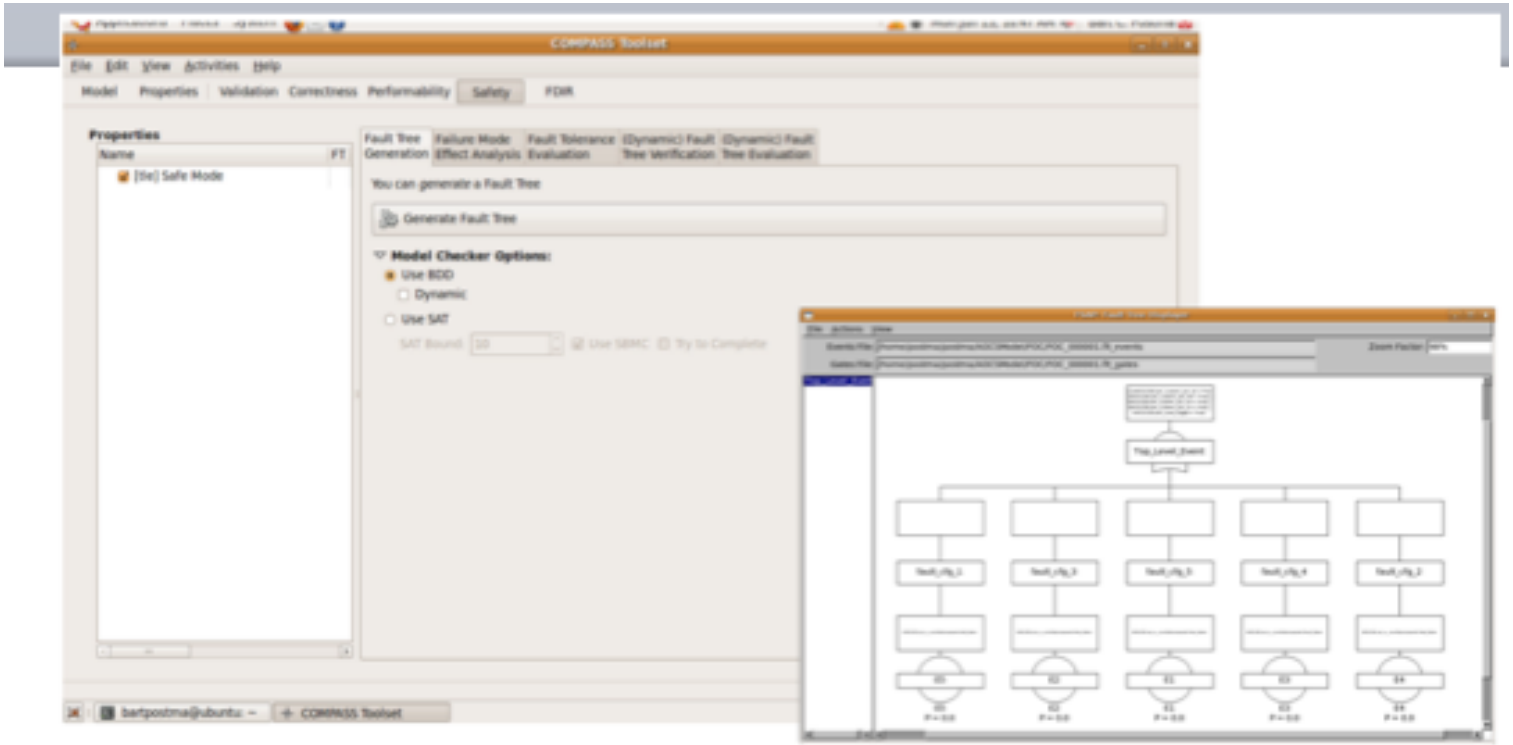
Performability



Diagnosis Constraints

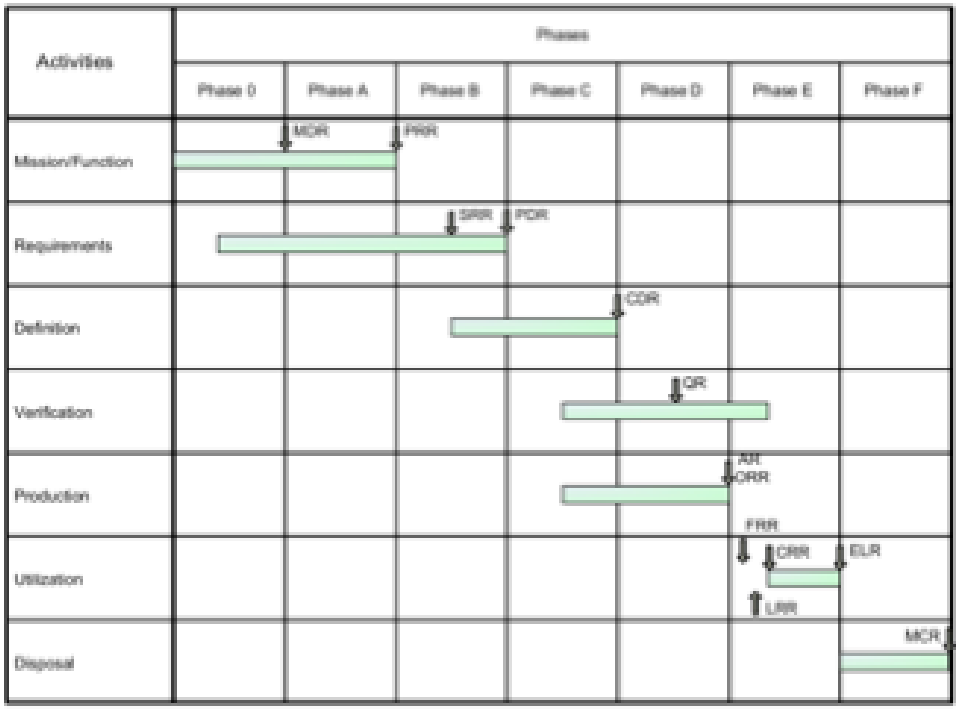
- Push-button operation, no tweaking of parameters.
- General good performance, handles large state spaces.
- Yet, need theoretic improvements to handle more.

Lesson 3: Industry-Ready Automated FMEA/FTA



- FTA/FMEA currently done manually. Takes months.
- Fast automated generation from model avoids
  - inconsistencies, and
  - missing causal links.

Lesson 4: Integration of Formal Methods



- Formal semantics with discrete, real-time, hybrid and probabilistic behavior creates coherence.
- More V&V analyses on the *same* formal semantics.
- Need progress measure on analyses for accounting.

Outline

Related Work

Formal AADL semantics

- Component-based semantics using BIP [Sifakis *et al.*, 2008]
- Arcade: AADL error annex [Stoelinga *et al.*, 2007]
- GSPN semantics [Kanoun *et al.*, 2007]
- .....

Related Work

Formal AADL semantics

- Component-based semantics using BIP [Sifakis *et al.*, 2008]
- Arcade: AADL error annex [Stoelinga *et al.*, 2007]
- GSPN semantics [Kanoun *et al.*, 2007]
- .....

AADL Analysis Tools

- AADL2BIP tool (simulation, deadlock detection) [Sifakis *et al.*, 2008]
- ADeS simulator www.axlor.fr
- Real-time scheduling tools Cheddar, Furness
- .....

Epilogue

Achievements

- Component-based modeling framework based on AADL
- Novelties: dynamic reconfiguration, hybridity, error modeling, ...
- Automated correctness, safety, and performability analysis
- Industrial evaluation by third-party company showed maturity

Trustworthy aerospace design = AADL modeling + analysis

Epilogue

Achievements

- Component-based modeling framework based on AADL
- Novelties: dynamic reconfiguration, hybridity, error modeling, ...
- Automated correctness, safety, and performability analysis
- Industrial evaluation by third-party company showed maturity

Trustworthy aerospace design = AADL modeling + analysis

Further information

- General approach (Yushstein et. al, IEEE SMC-IT 2011)
  (Bozzano et. al, ACES-MB 2009)
  (Bozzano et. al, SAFECOMP 2009)
- AADL model checker (Bozzano et. al, CAV 2010)
- Thales case studies (Bozzano et. al, ERTS^2 2010)
- ESA satellite case study (Esteve et. al, ICSE 2012)
- Tool download at `http://compass.informatik.rwth-aachen.de/`