Safety, Dependability and Performance Analysis of Aerospace Systems using the COMPASS Toolset

Thomas Noll (noll@cs.rwth-aachen.de)

6th FMF Formal Methods Day
LAAS-CNRS, Toulouse, France; January 26, 2015

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Overview

Outline

Fault-Tolerant Space System Architectures

Space system requirements

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

Fault-Tolerant Space System Architectures

Space system requirements

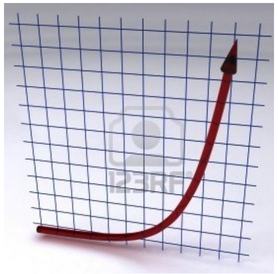
- Must offer service without interruption for a very long time – typically years or decades
- Failures are costly and may severely damage reputations:
  - Ariane 5 crash in 1996 due to arithmetic overflow
  - Launch failure of Phobos-Grunt sample return mission
- “Five nines” (99.999 %) dependability 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

Motivation

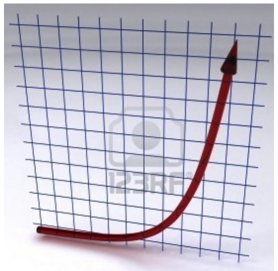
Challenges for Verification & Validation



Software size grows exponentially

- Apollo (1970) 8,500 LOC
- Space Shuttle (1980) 470,000 LOC
- ISS (1995) 1,000,000 LOC

Challenges for Verification & Validation



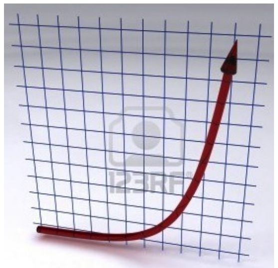
Software size grows exponentially

• Apollo (1970)	8,500 LOC
• Space Shuttle (1980)	470,000 LOC
• ISS (1995)	1,000,000 LOC

More continuous verification & validation to manage risks/budgets/planning

- Requirements analysis
- HW/SW co-testing
- In-orbit testing, etc.

Challenges for Verification & Validation



Software size grows exponentially

- Apollo (1970) 8,500 LOC
- Space Shuttle (1980) 470,000 LOC
- ISS (1995) 1,000,000 LOC

More continuous verification & validation to manage risks/budgets/planning

- Requirements analysis
- HW/SW co-testing
- In-orbit testing, etc.

System-software engineering lacks coherence

- HW/SW verified in isolation and with exaggerated mutual assumptions
- Safety & dependability analysis separated from HW/SW models
- Manifold modelling formalisms for real-time/hybrid/risk aspects

Outline

COMPASS Project Overview

COrrectness, Modelling and Performance of AeroSpace Systems

The COMPASS mission

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

COMPASS Project Overview

COrrectness, Modelling and Performance of AeroSpace Systems

The COMPASS mission

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

Derived objectives

1. Modelling formalism: SLIM
   (System-Level Integrated Modelling Language; variant of AADL)
2. Verification methodology based on state-of-the-art formal methods
3. Toolset supporting the analysis of SLIM models
4. Evaluation on industrial-size case studies from aerospace domain

COMPASS Project Partners

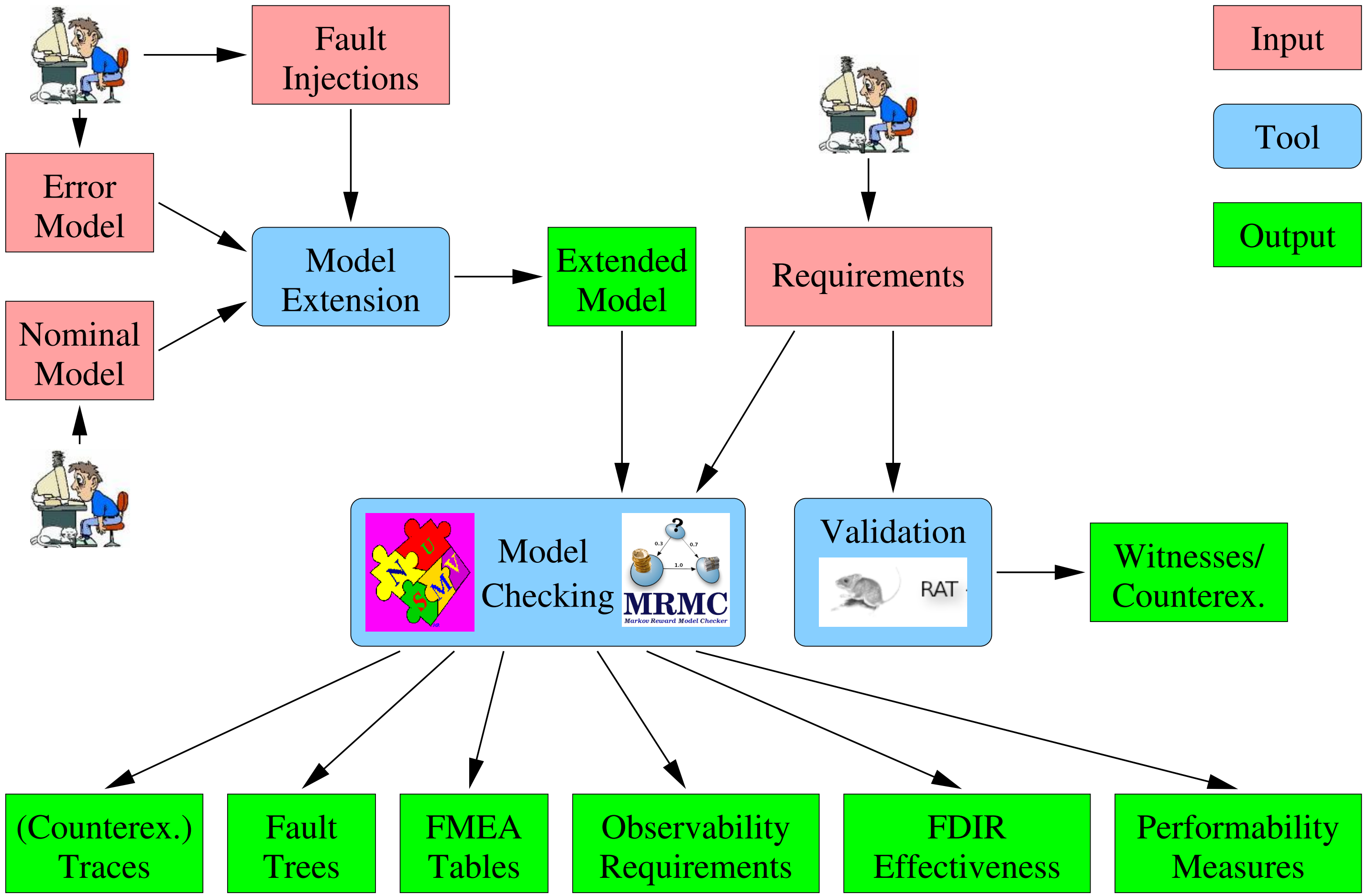
Consortium

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

Funding & supervision

- European Space Agency

COMPASS Methodology



Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Outline

System Specifications

SLIM: Specification Language Based on AADL

AADL (since 1989 by SAE)

Architecture modelling language for safety-critical systems featuring:

- Components and hierarchy
- HW (processors, devices, buses, etc.)
- SW (processes, threads, etc.)
- Modes and mode transitions
- Event/data port communication
- Dynamic reconfiguration

System Specifications

SLIM: Specification Language Based on AADL

AADL (since 1989 by SAE)

Architecture modelling language for safety-critical systems featuring:

- Components and hierarchy
- HW (processors, devices, buses, etc.)
- SW (processes, threads, etc.)
- Modes and mode transitions
- Event/data port communication
- Dynamic reconfiguration

SLIM (since 2008 by us)

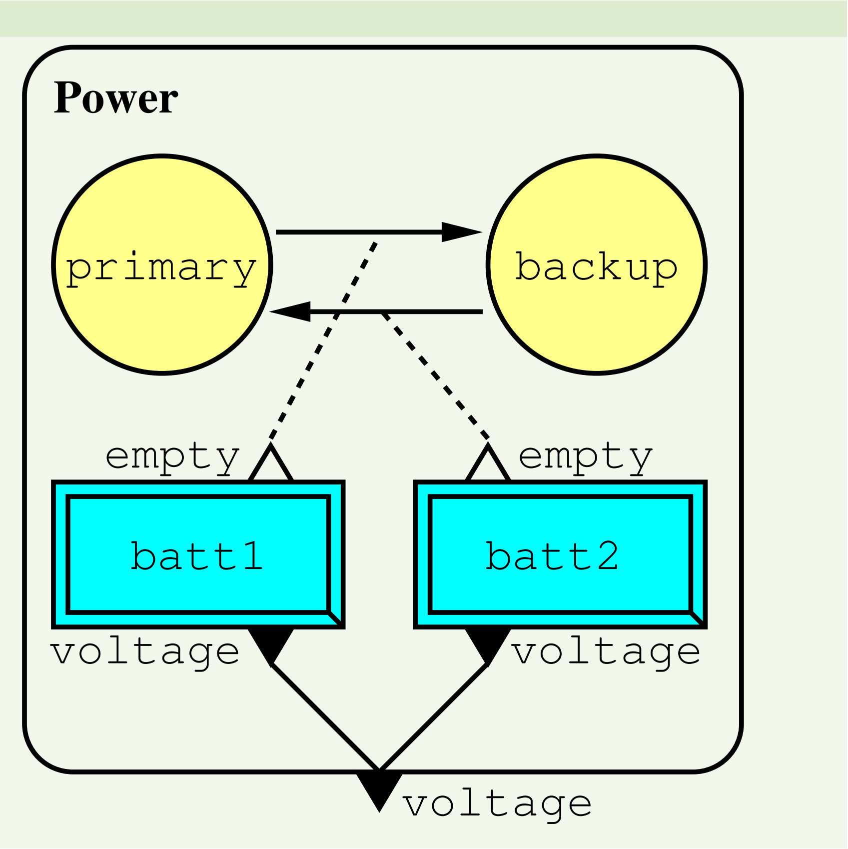
System-level integrated modelling language for space systems featuring:

- Major part of AADL V1.0, and
- Functional, real-time and hybrid behaviour
- Error events and error states (Error Model Annex)
- Formal semantics

Running 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



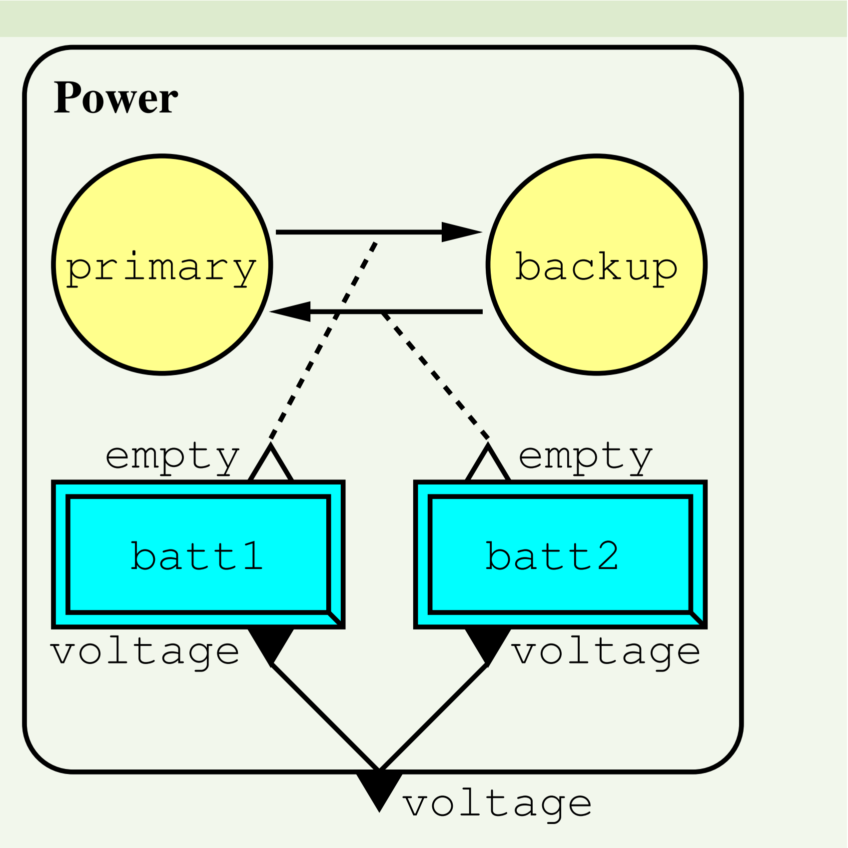
Running Example: Redundant Power System

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 see:

- hybrid behaviour of the batteries
- composition of the power system
- interweaving of errors

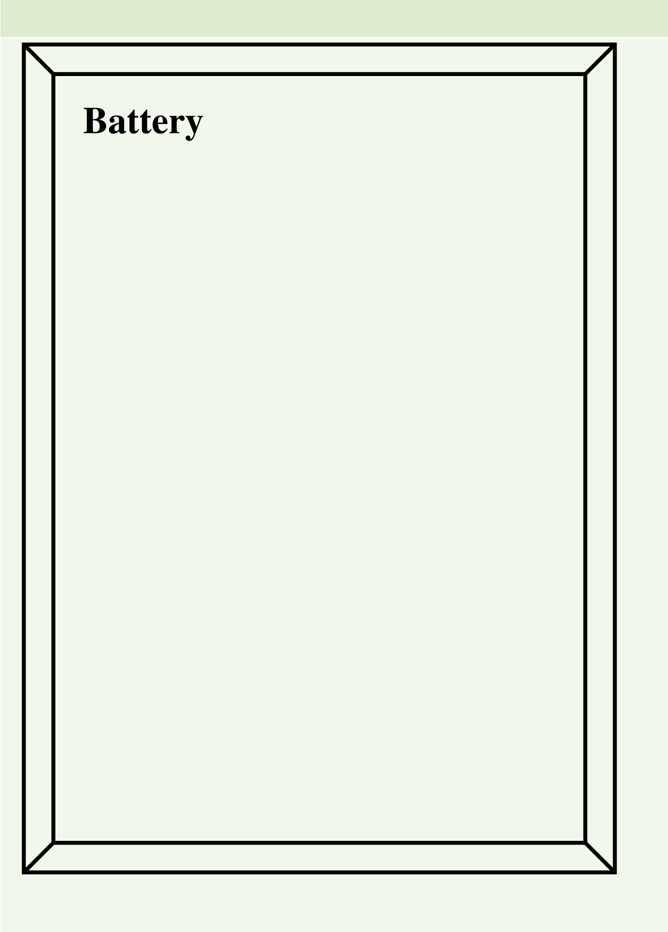


Software Modeling and Verification Chair

Modelling a Battery

Component type and implementation:

```
device type Battery


end Battery;

device implementation Battery.Imp






end Battery.Imp;
```

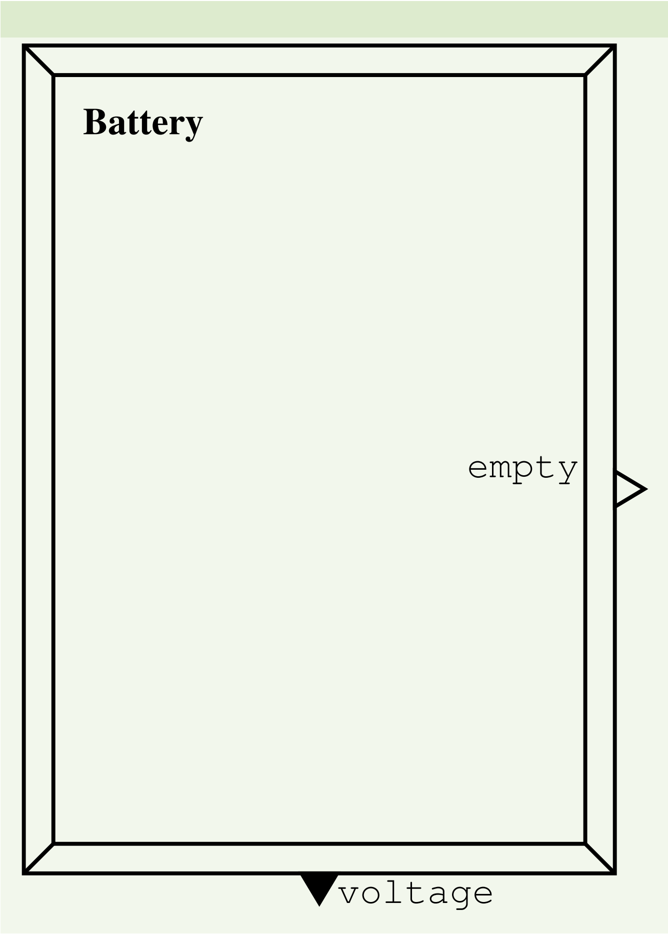


System Specifications

Modelling a Battery

Type defines the interface:

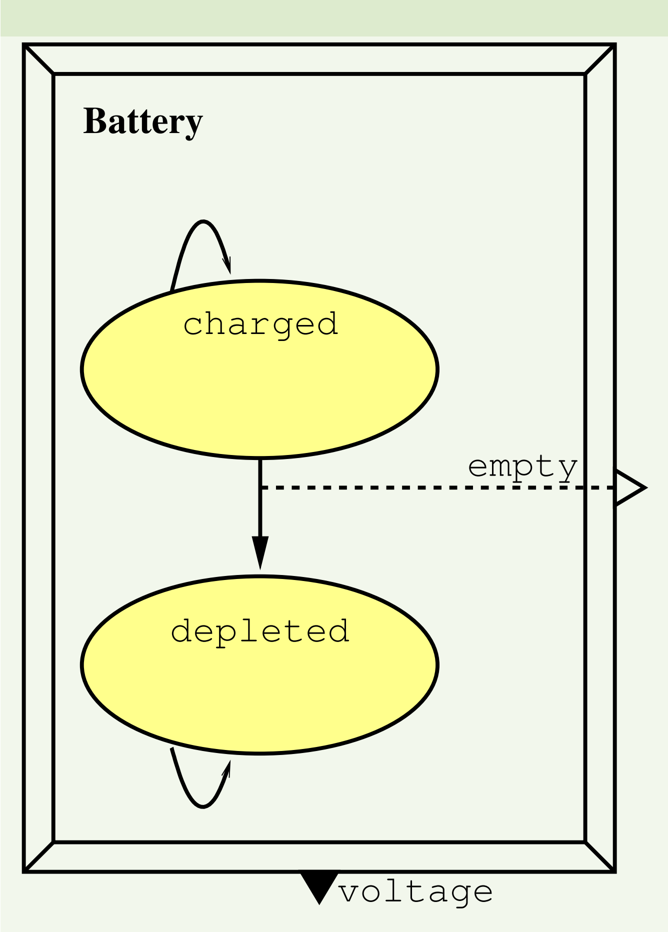
```
device type Battery
  features
    empty: out event port;
    voltage: out data port real default 6.0;
end Battery;

device implementation Battery.Imp




end Battery.Imp;
```



Modelling a Battery

Adding mode behaviour:

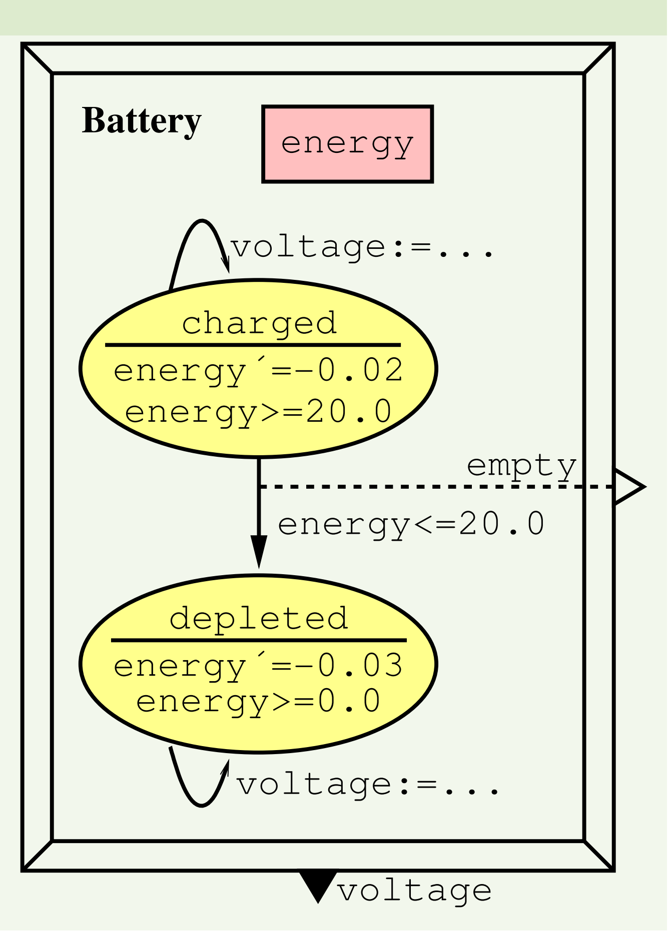
```
device type Battery
  features
    empty: out event port;
    voltage: out data port real default 6.0;
end Battery;

device implementation Battery.Imp


  modes
    charged: initial mode

    depleted: mode

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



Modelling a Battery

Adding hybrid behaviour:

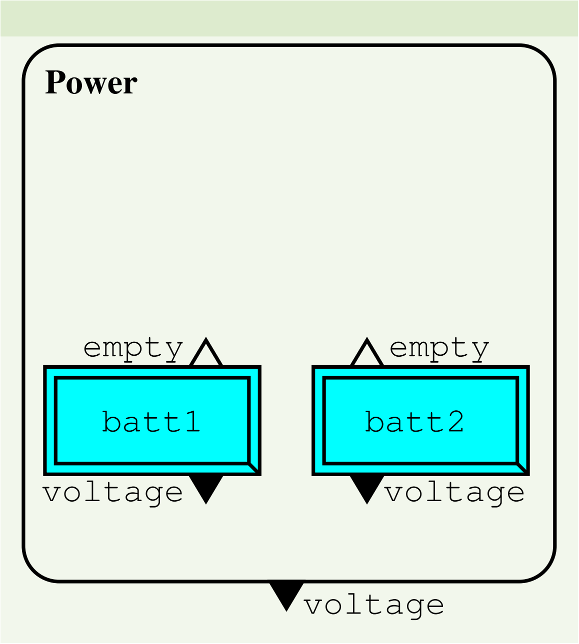
```
device type Battery
  features
    empty: out event port;
    voltage: out data port real default 6.0;
end Battery;

device implementation Battery.Imp
  subcomponents
    energy: data continuous default 1.0;
  modes
    charged: initial mode
      while energy'=-0.02 and energy>=20.0;
    depleted: mode
      while energy'=-0.03 and energy>=0.0;
  transitions
    charged -[then voltage:=2.0*energy+4.0]-> charged;
    charged -[empty when energy<=20.0]-> depleted;
    depleted -[then voltage:=2.0*energy+4.0]-> depleted;
end Battery.Imp;
```



Modelling the Redundant Power System

Power system with battery subcomponents:

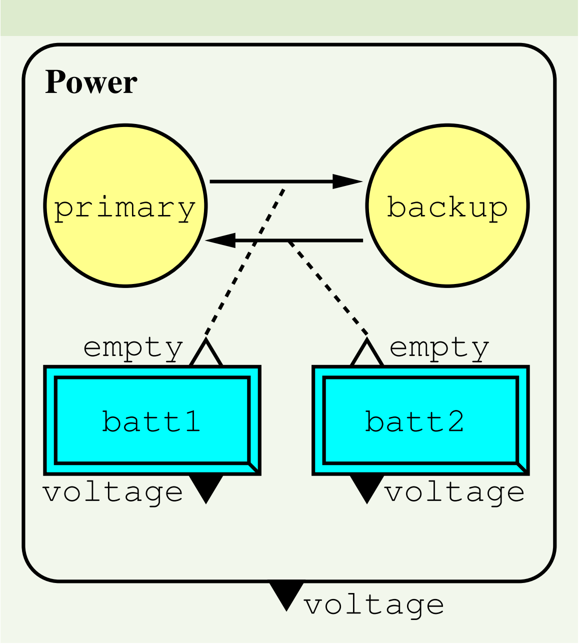
```
system Power
  features
    voltage: out data port real;
end Power;

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




end Power.Imp;
```



Modelling 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;
```

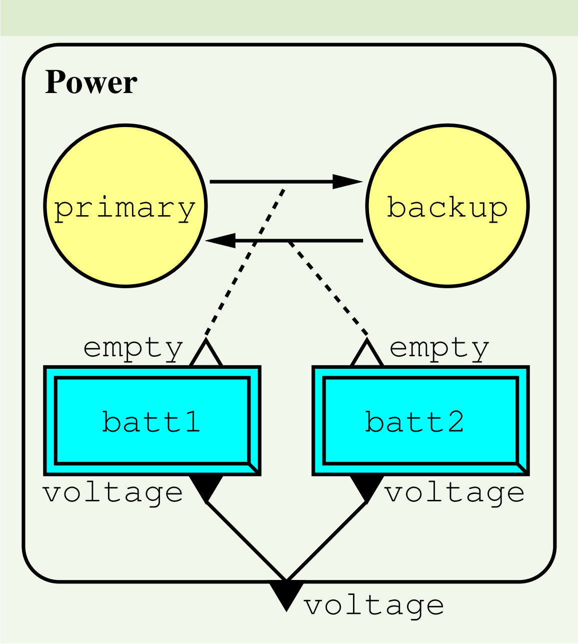


System Specifications

Modelling the Redundant Power System

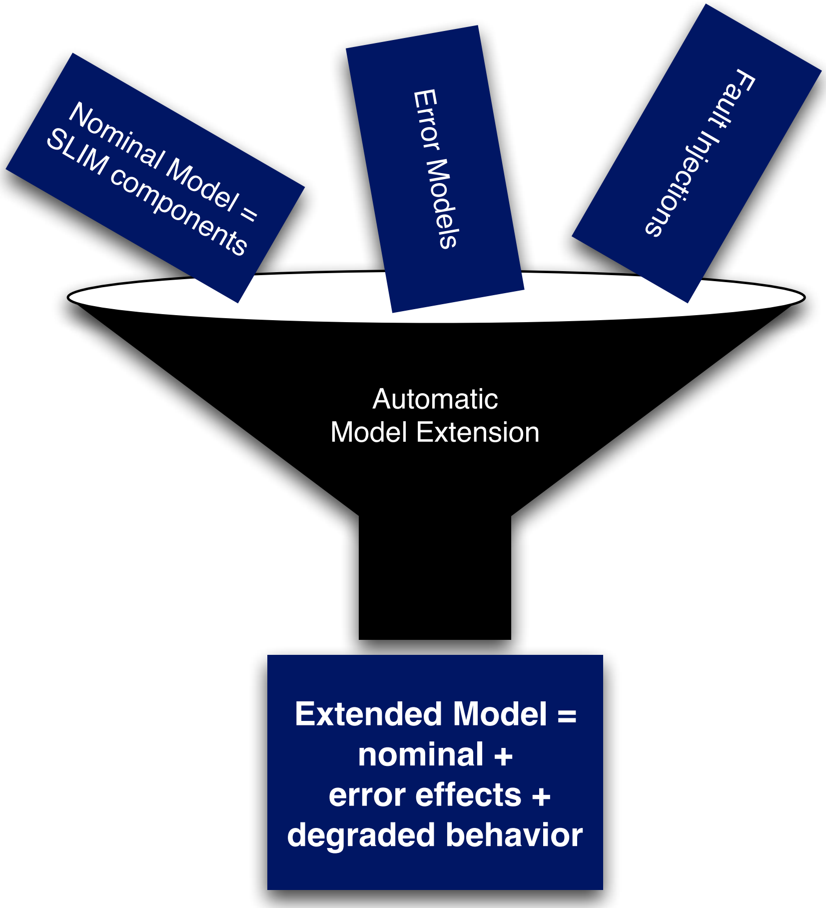
Adding port connections:

```
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;
```



Outline

Integrating Erroneous and Nominal Behaviour



Error Modelling

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

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

Error Modelling

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

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

Fault injection

An error model does not influence the nominal behaviour 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 Modelling

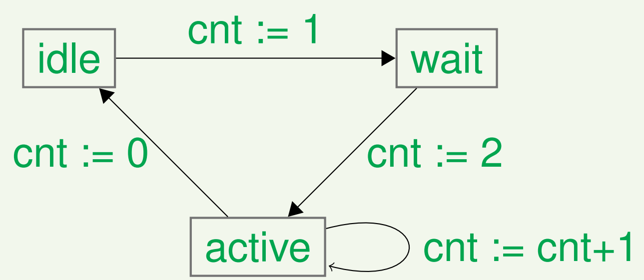
```
error model BatteryFailure
  features
    ok: initial state;
    dead:  error state;
    died:  out error propagation;
end BatteryFailure;

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

Fault injection

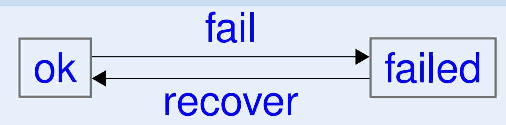
In error state `dead`, `voltage := 0`

Model Extension by Example

Nominal behaviour

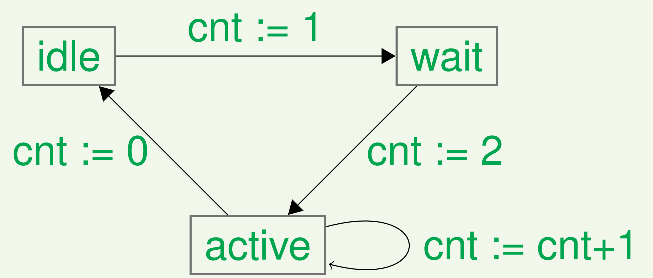


Error behaviour



Model Extension by Example

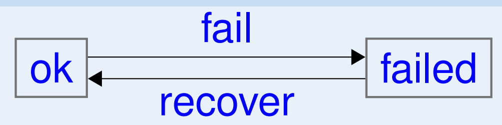
Nominal behaviour



Fault injection

failed: cnt := -1

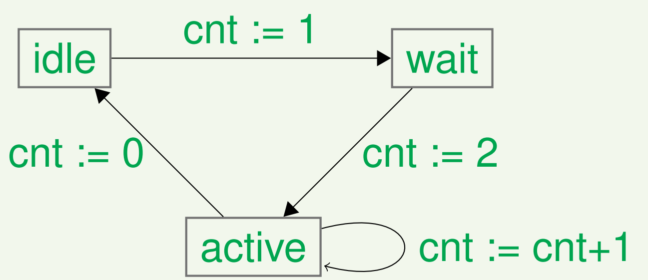
Error behaviour



Error Modelling

Model Extension by Example

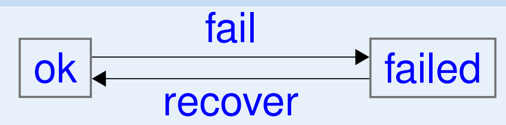
Nominal behaviour



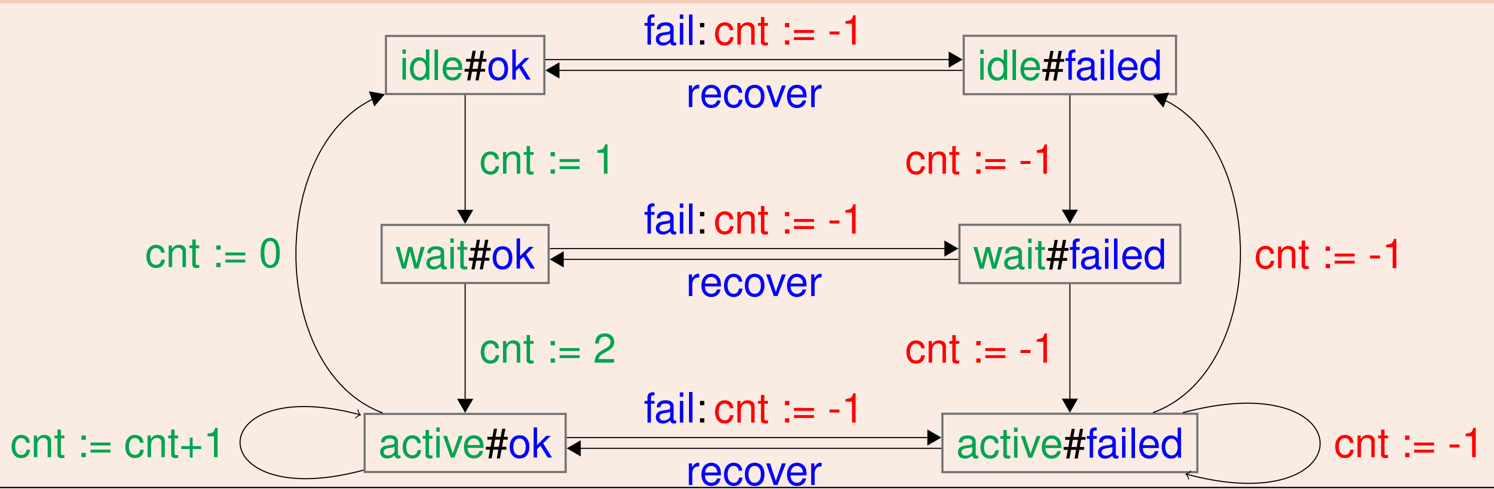
Fault injection

failed: cnt := -1

Error behaviour



Automatically extended model



Outline

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Requirements: Patterns, not Formulas!

Patterns

- The system shall have a behaviour where ϕ globally holds.
- The system shall have a behaviour 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!

Instantiated patterns

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

Requirements: Patterns, not Formulas!

Instantiated patterns

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

(by automatic transformation)

Logic

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

Requirements: Patterns, not Formulas!

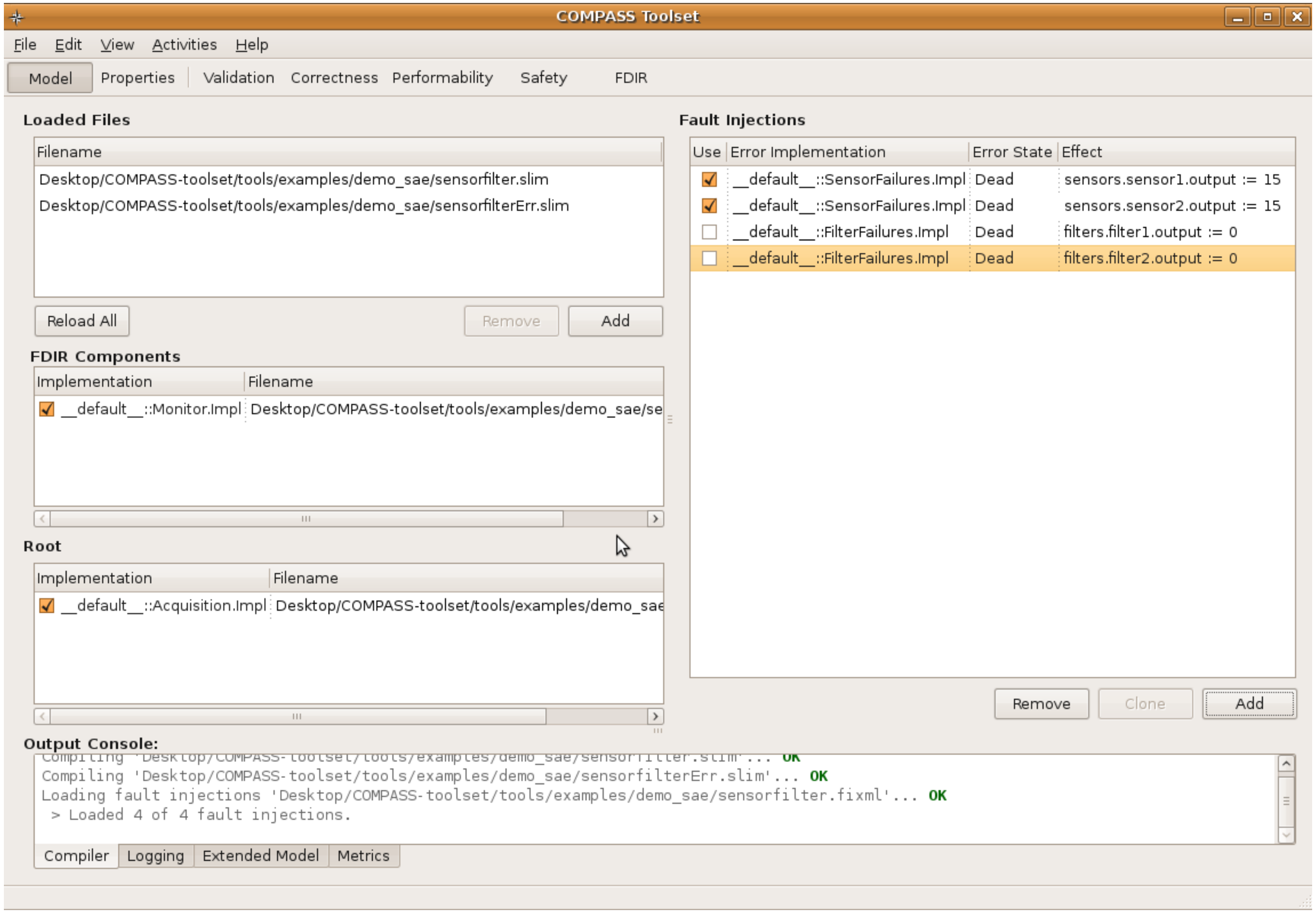
Logic

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

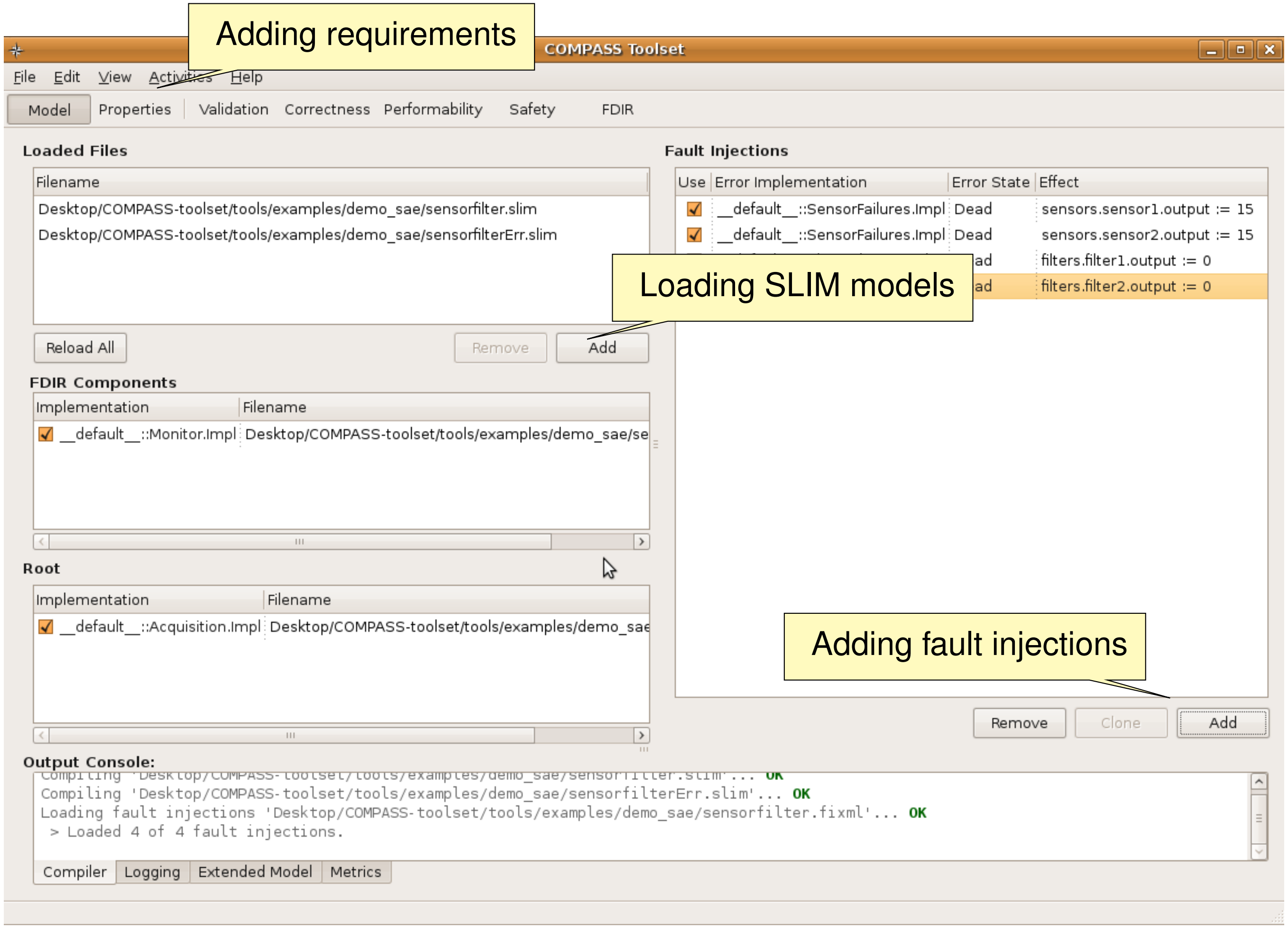
Implemented pattern systems

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]

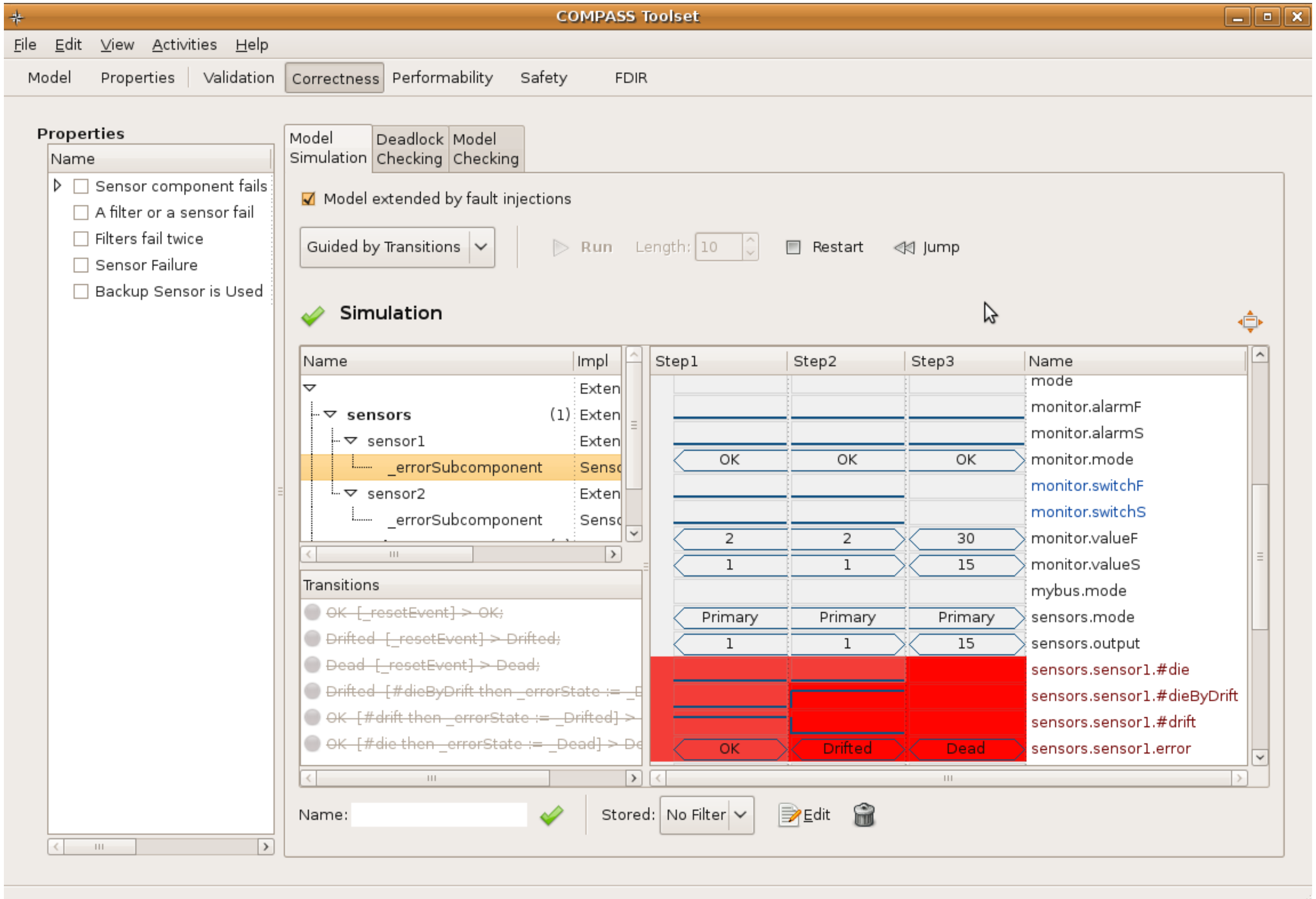
COMPASS Toolset: Main



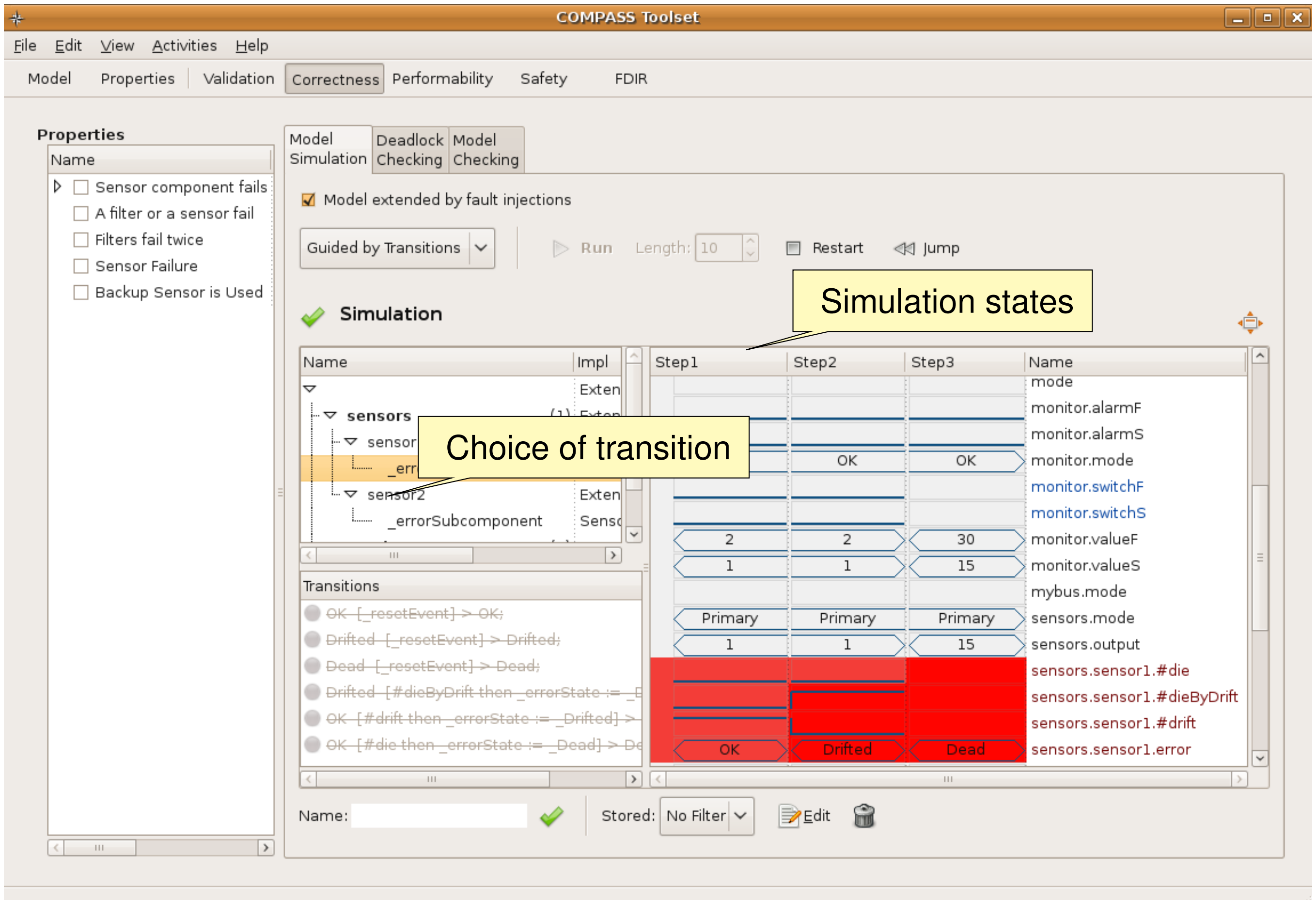
COMPASS Toolset: Main



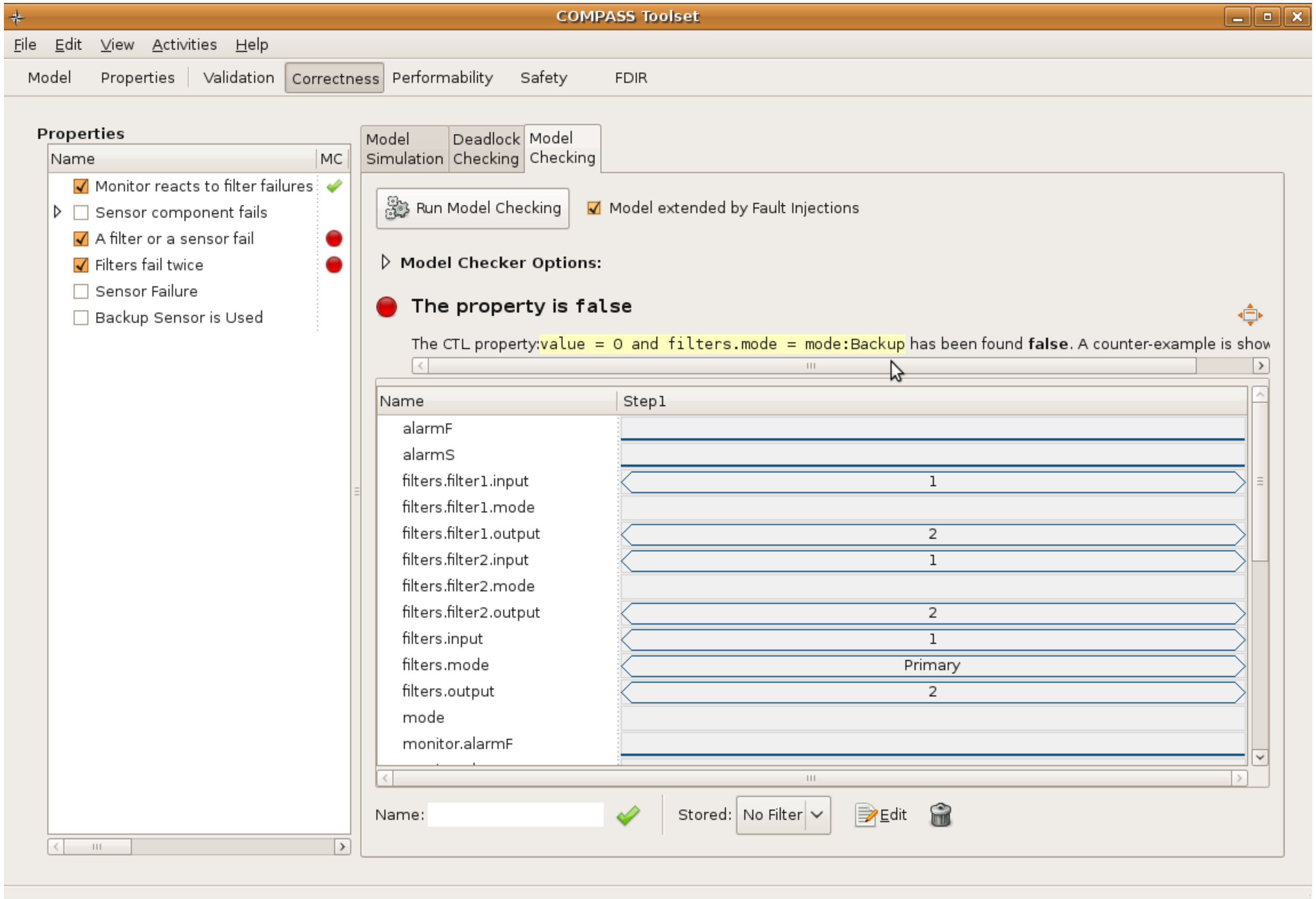
COMPASS Toolset: Simulation



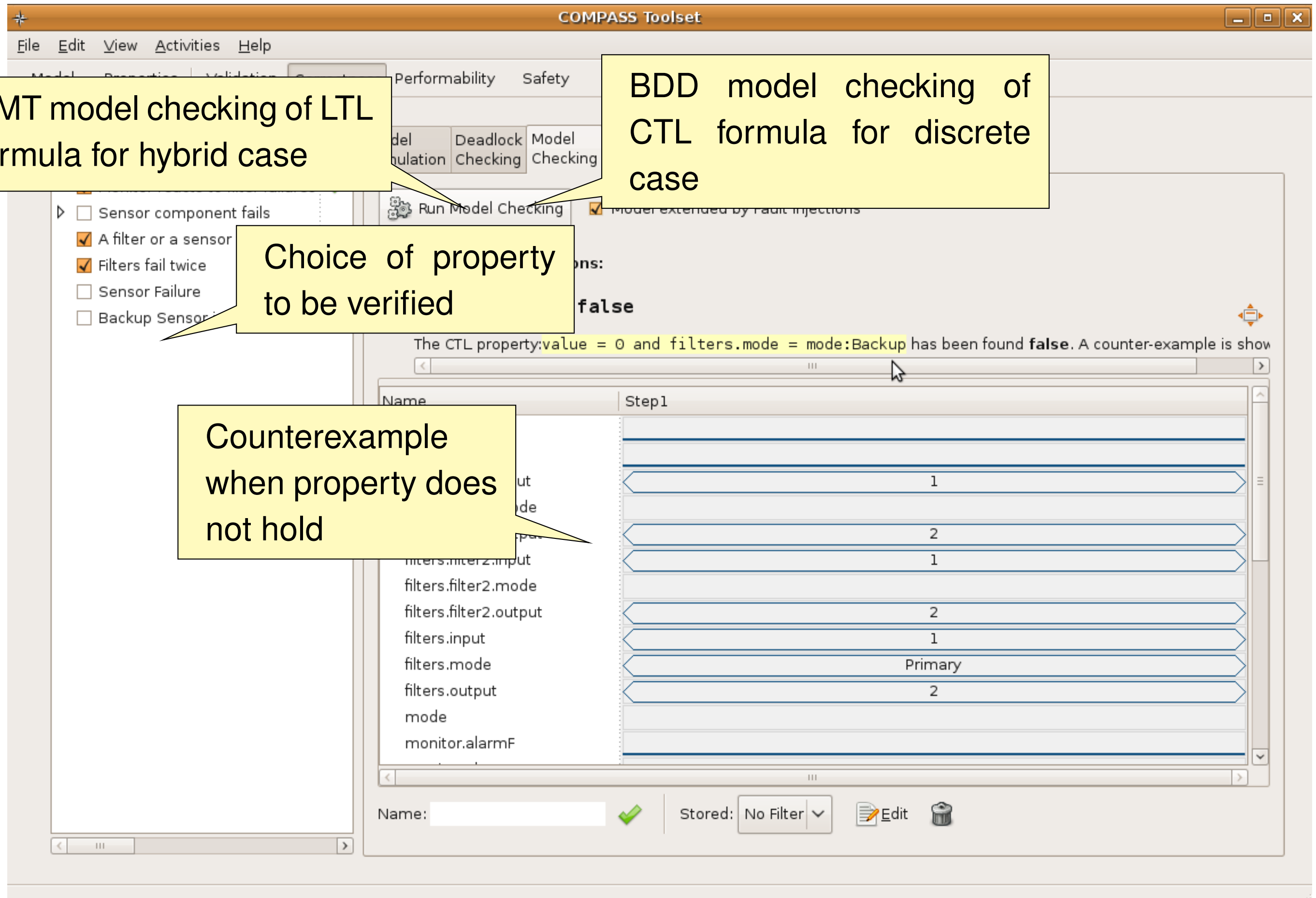
COMPASS Toolset: Simulation



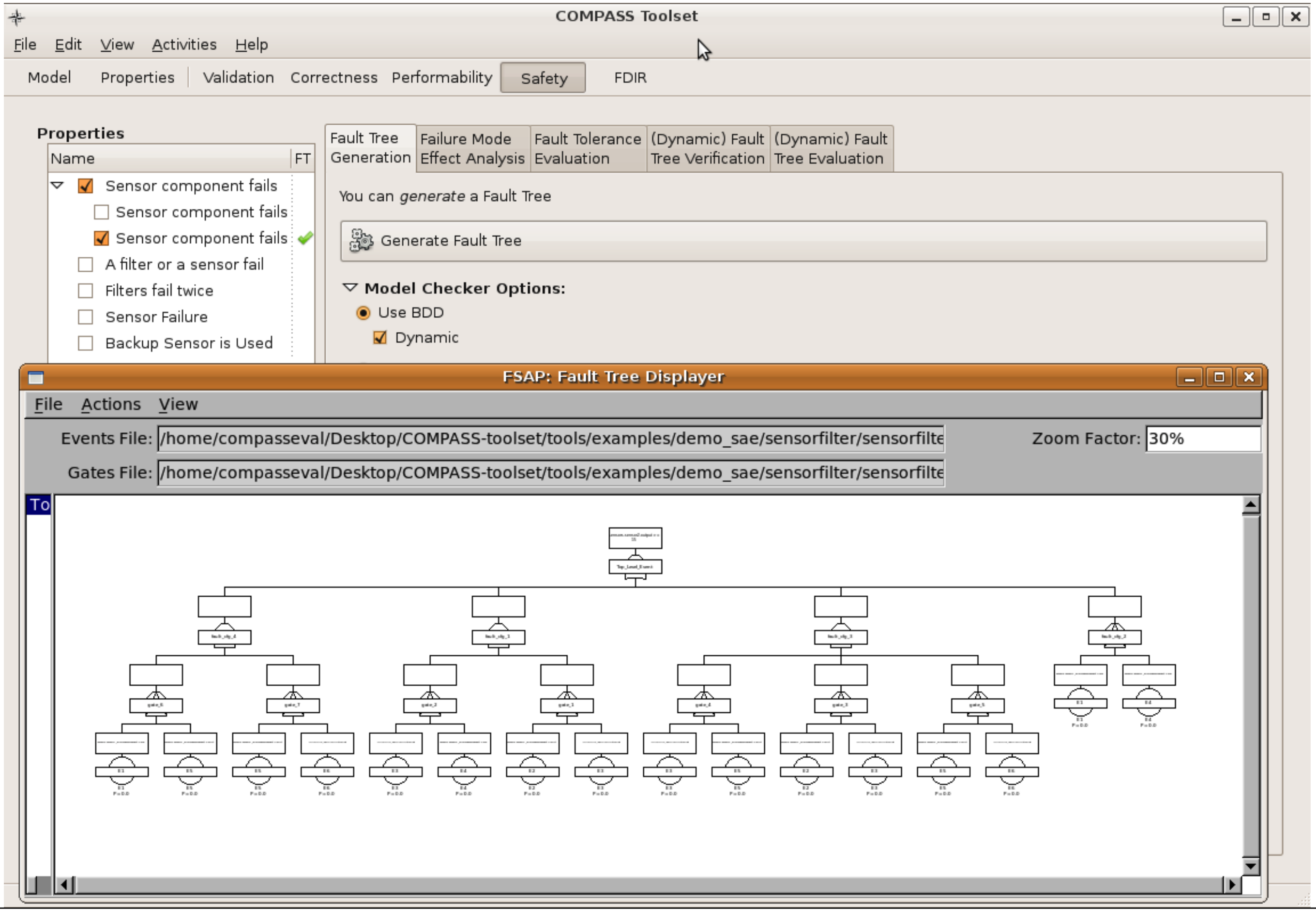
COMPASS Toolset: Model Checking



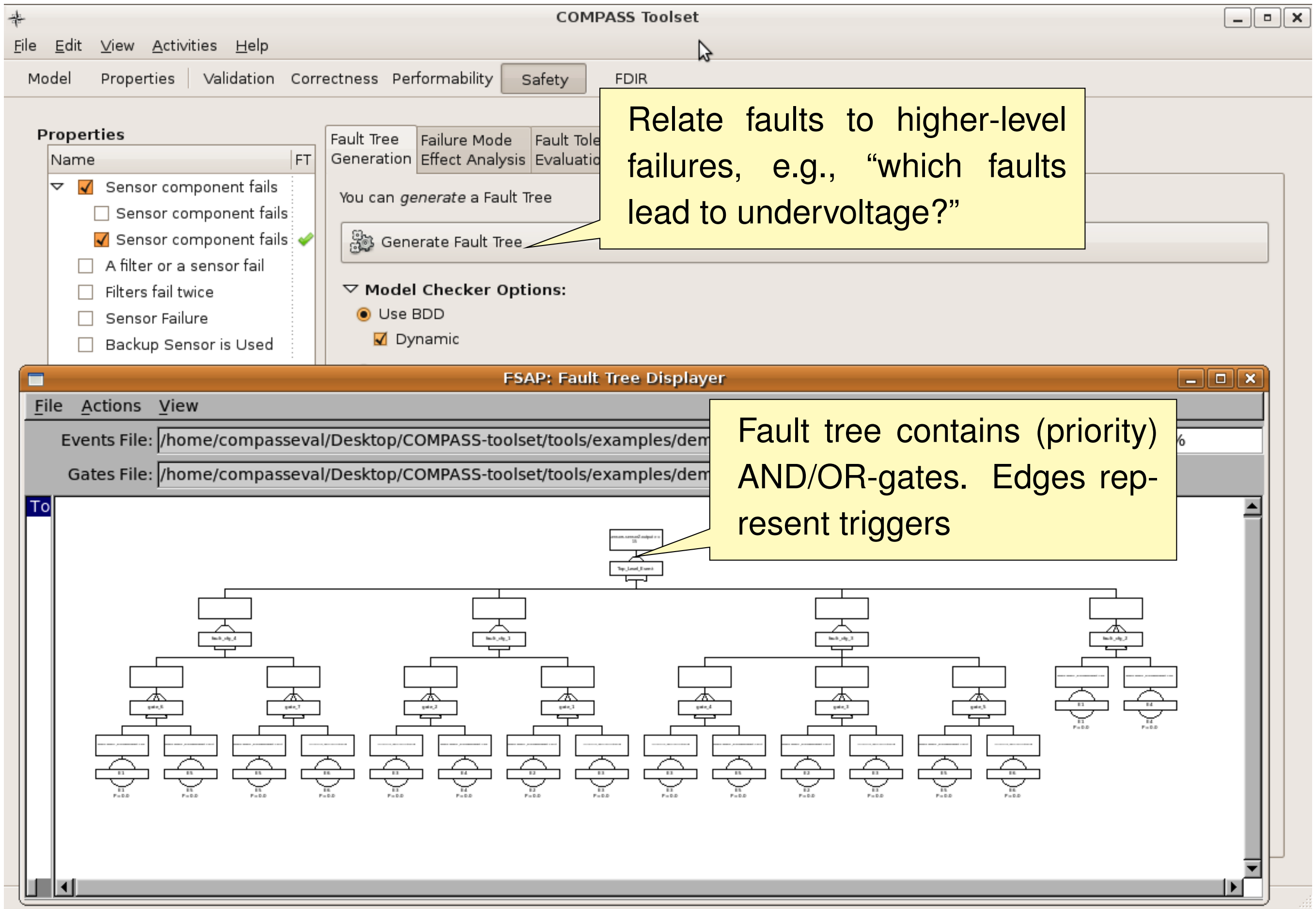
COMPASS Toolset: Model Checking



COMPASS Toolset: Fault Tree Analysis



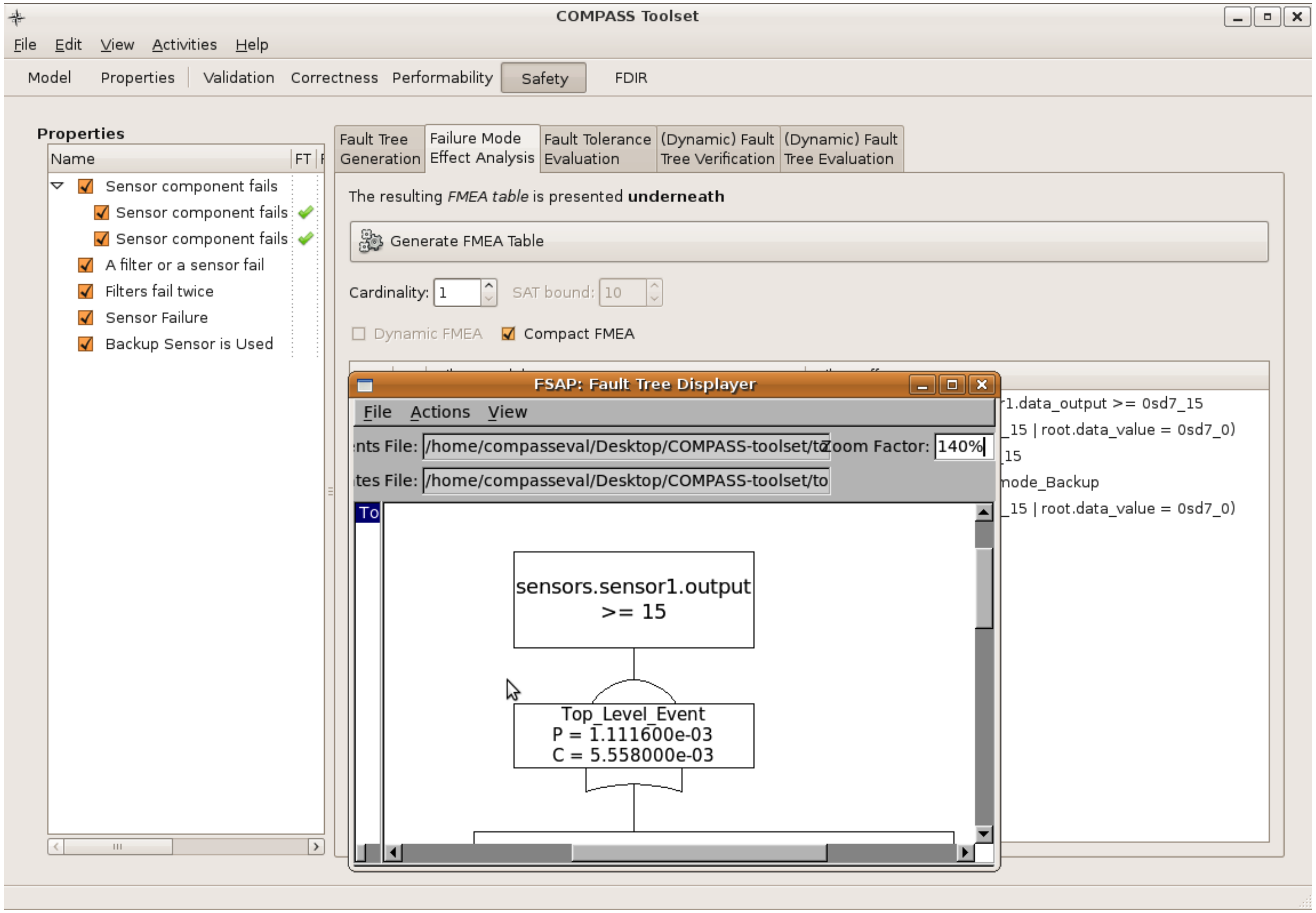
COMPASS Toolset: Fault Tree Analysis



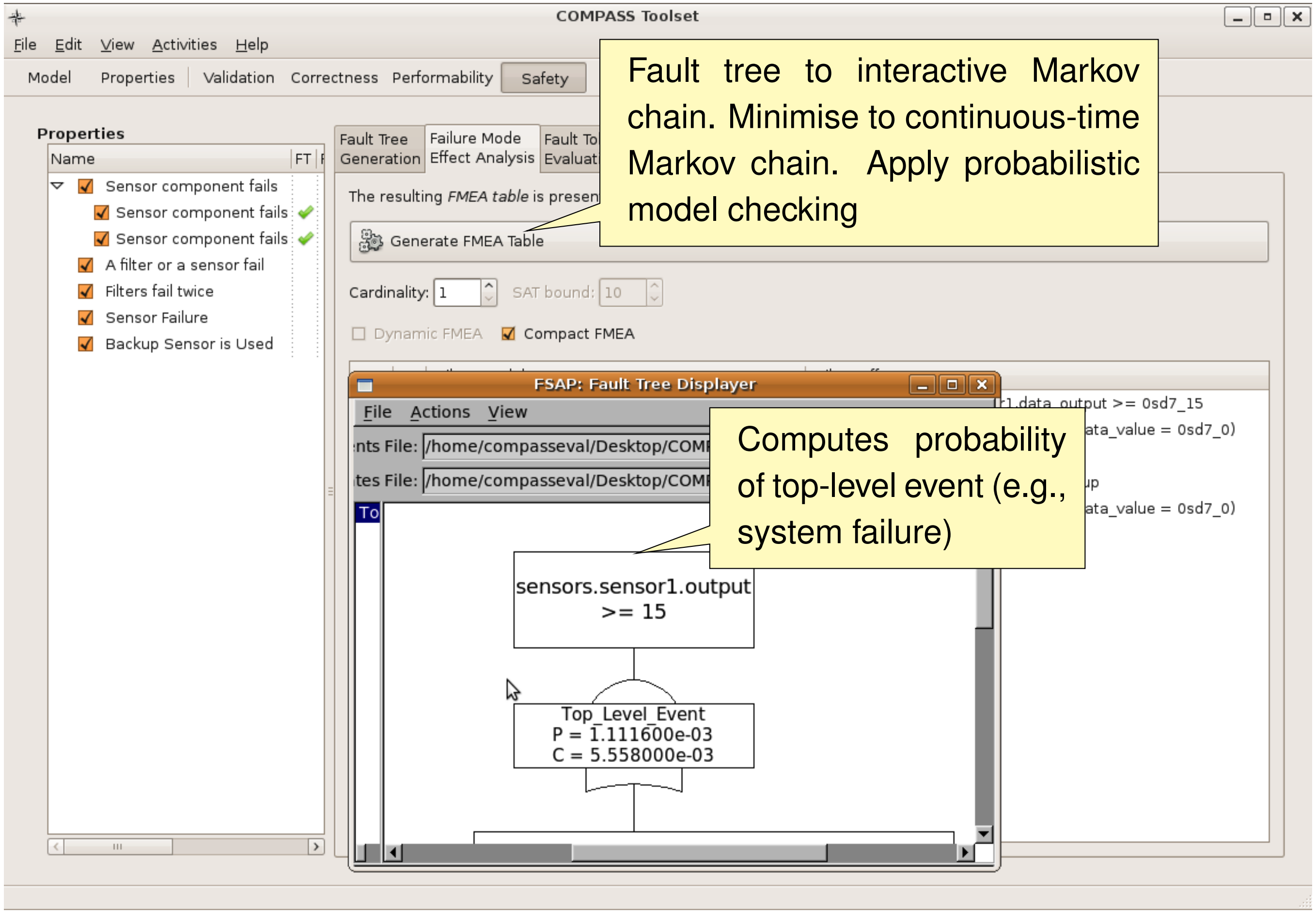
Relate faults to higher-level failures, e.g., "which faults lead to undervoltage?"

Fault tree contains (priority) AND/OR-gates. Edges represent triggers

COMPASS Toolset: Probabilistic Risk Assessment



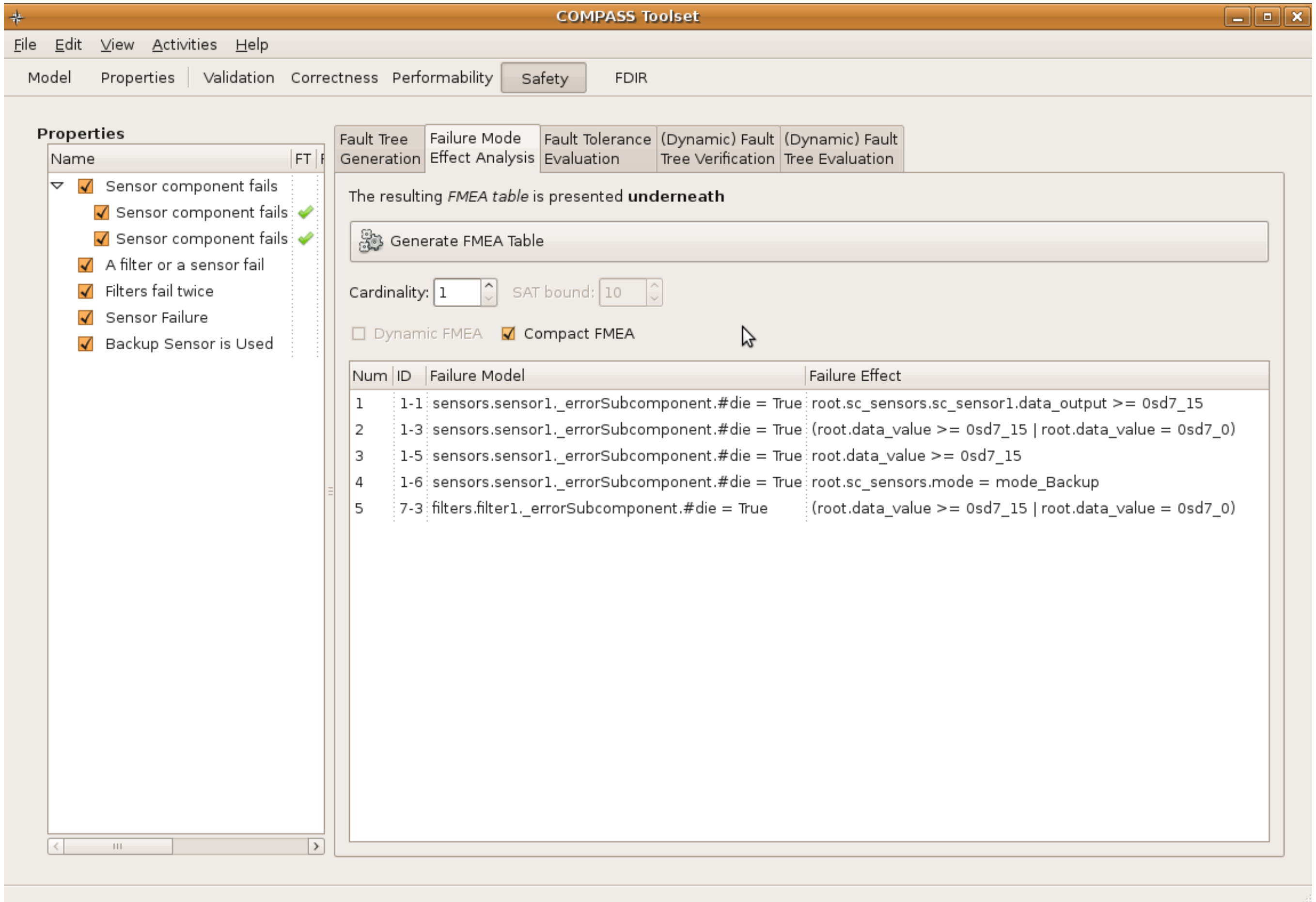
COMPASS Toolset: Probabilistic Risk Assessment



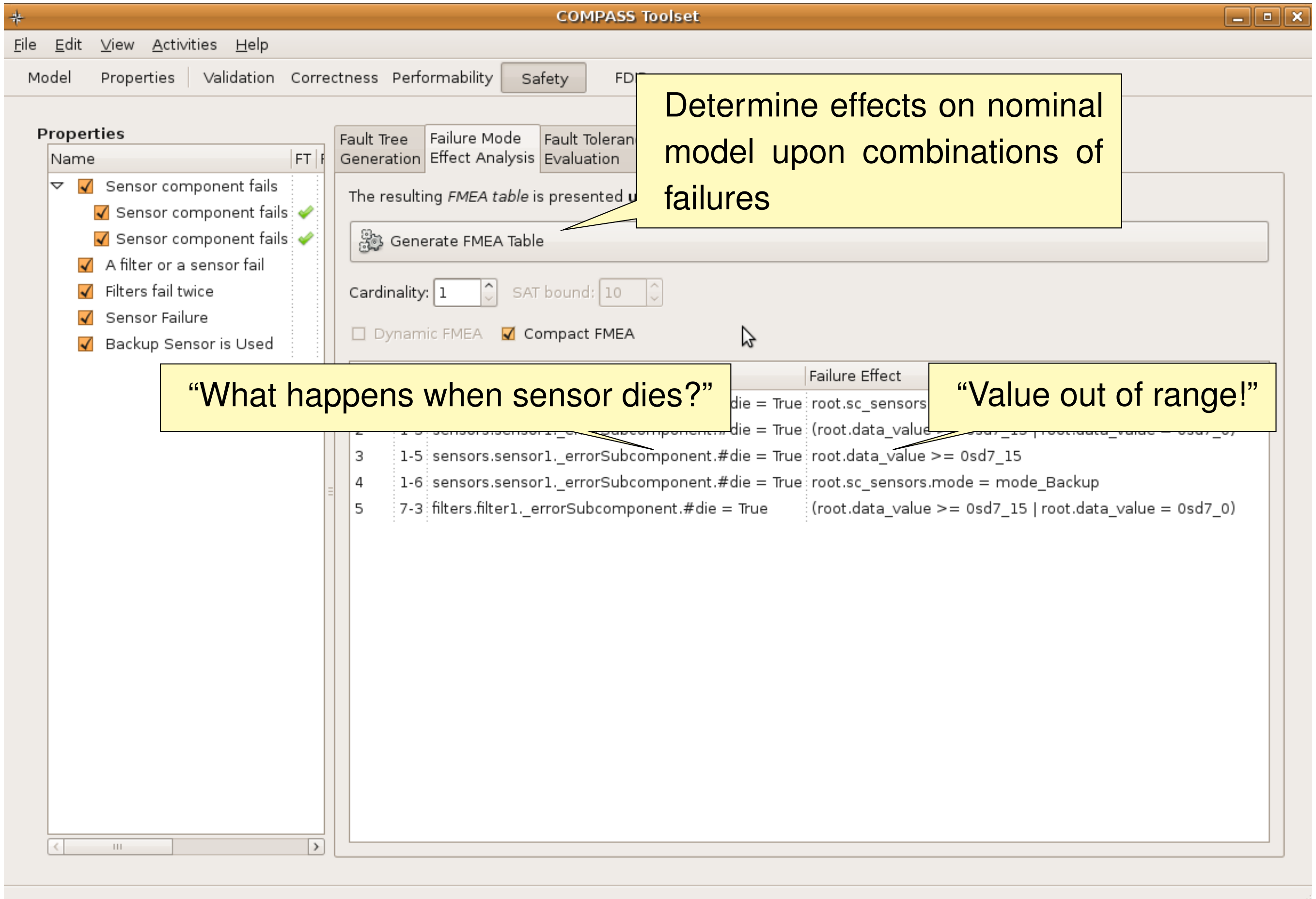
Fault tree to interactive Markov chain. Minimise to continuous-time Markov chain. Apply probabilistic model checking

Computes probability of top-level event (e.g., system failure)

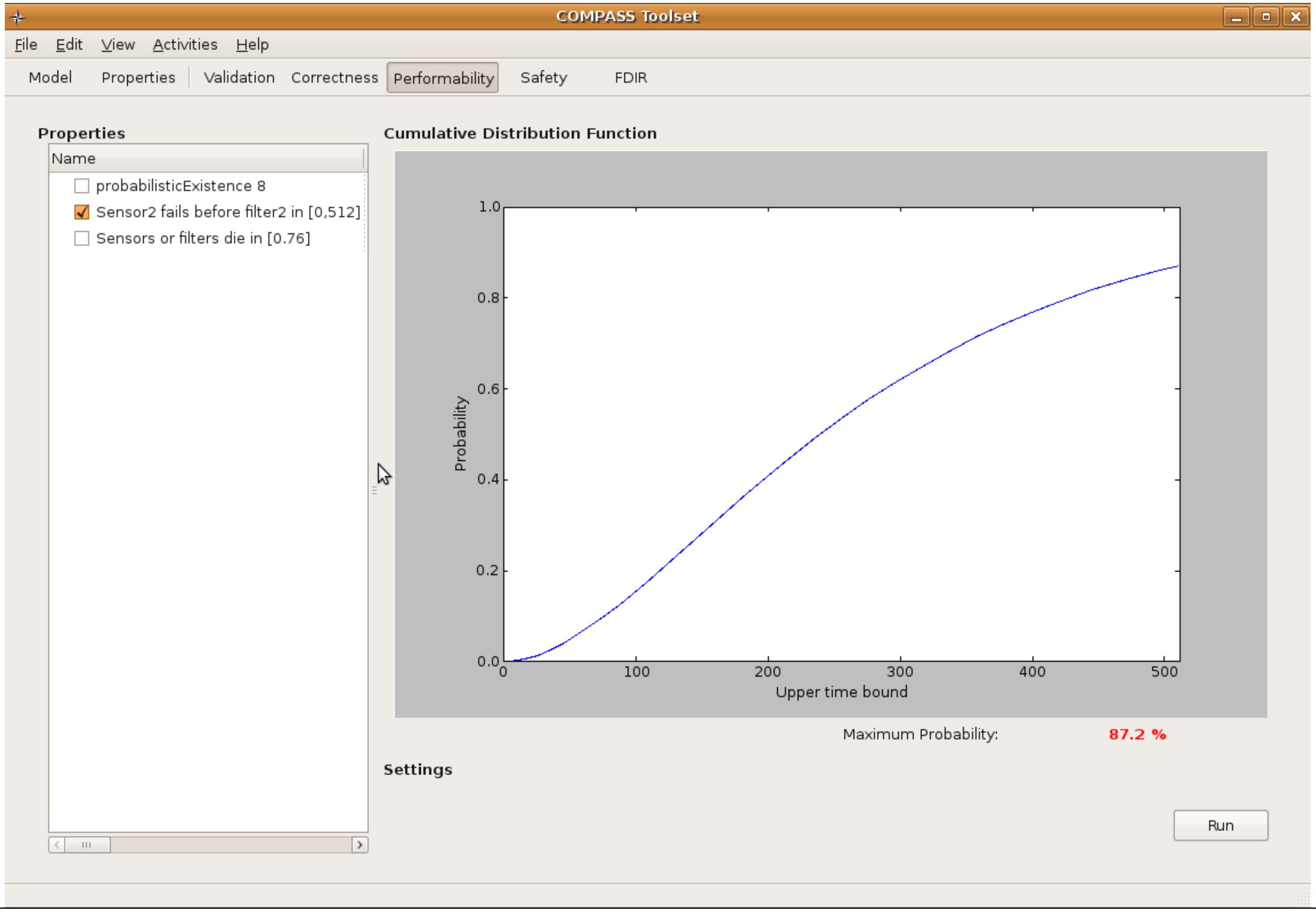
COMPASS Toolset: FMEA Analysis



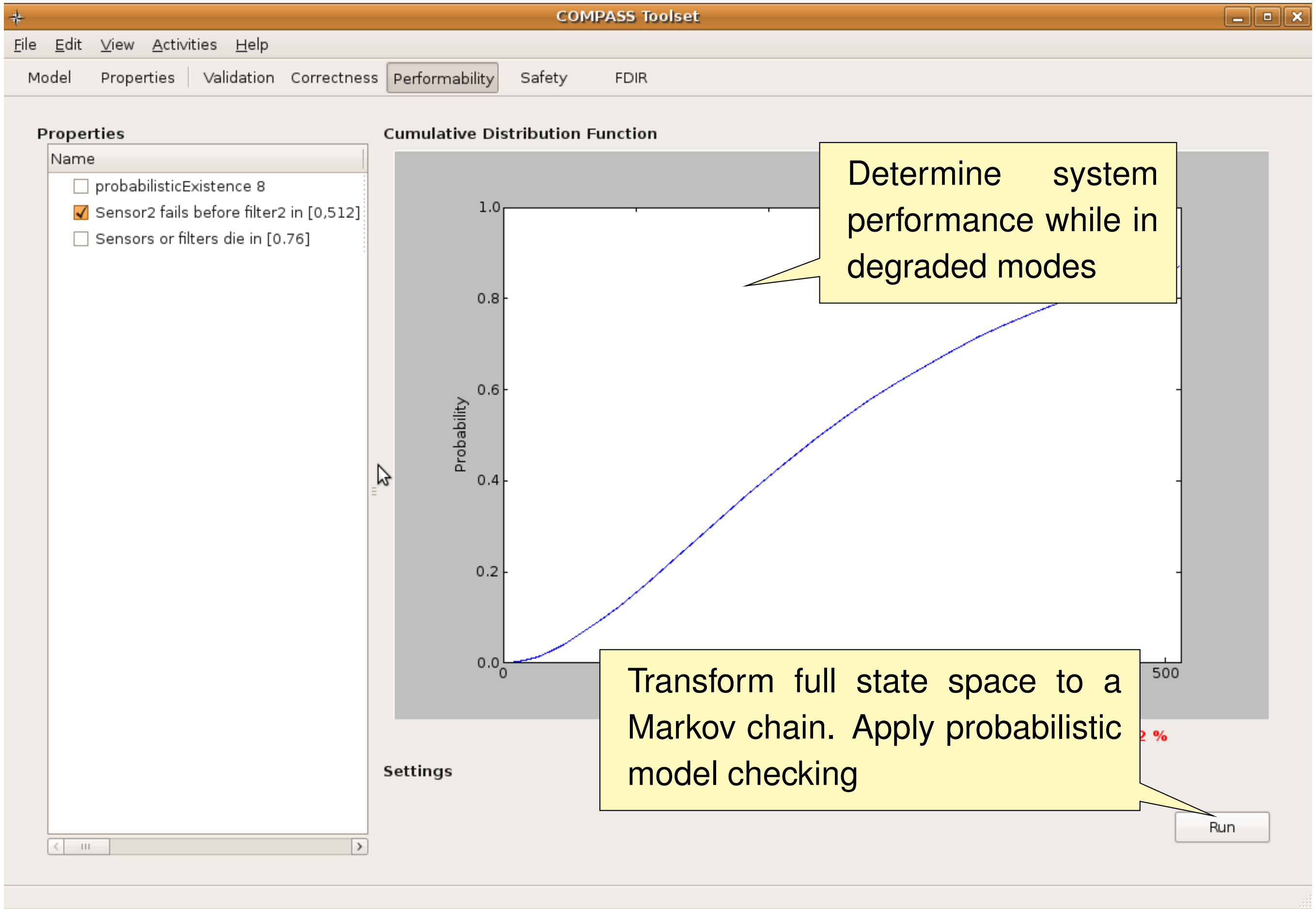
COMPASS Toolset: FMEA Analysis



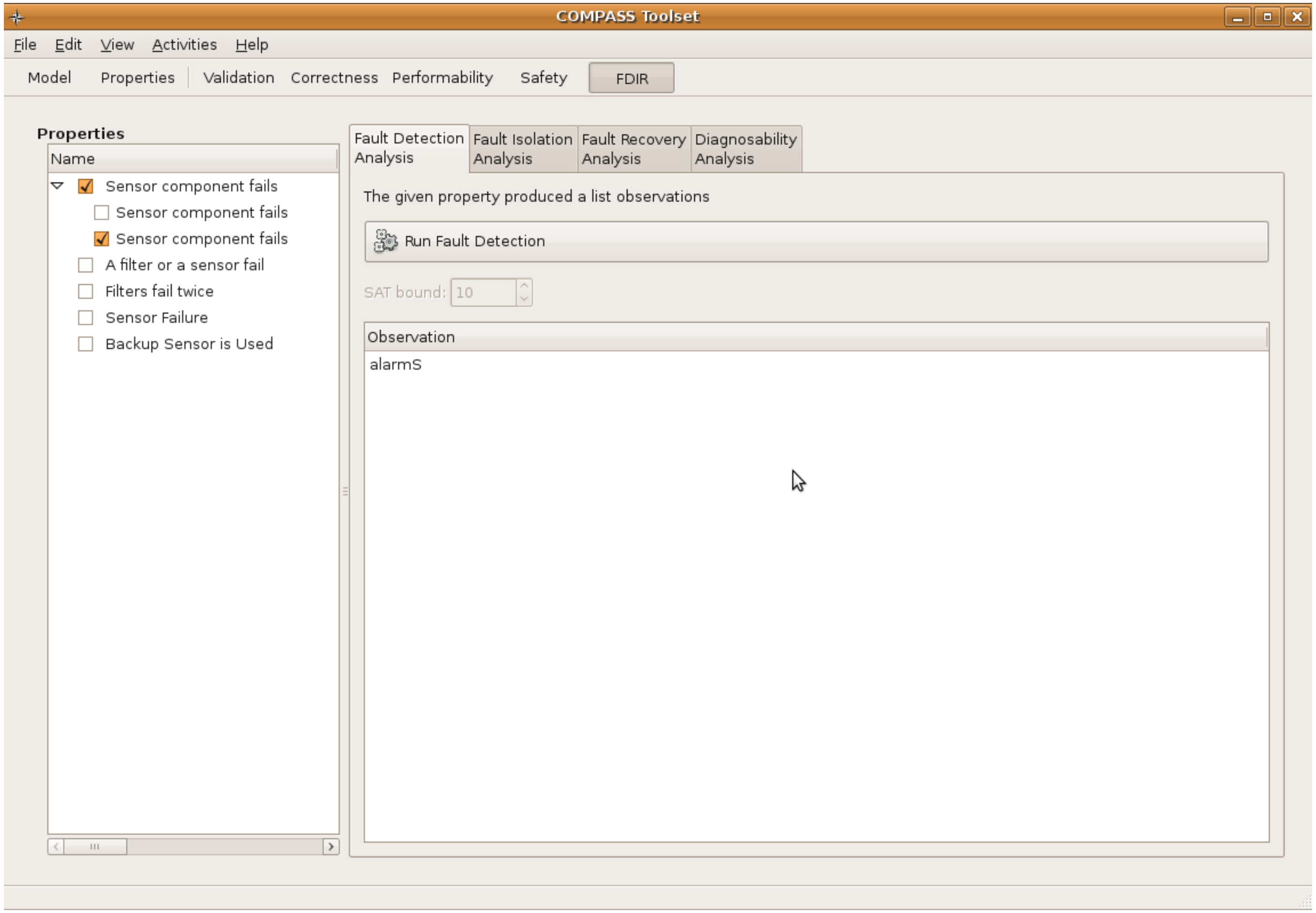
COMPASS Toolset: Performability



COMPASS Toolset: Performability



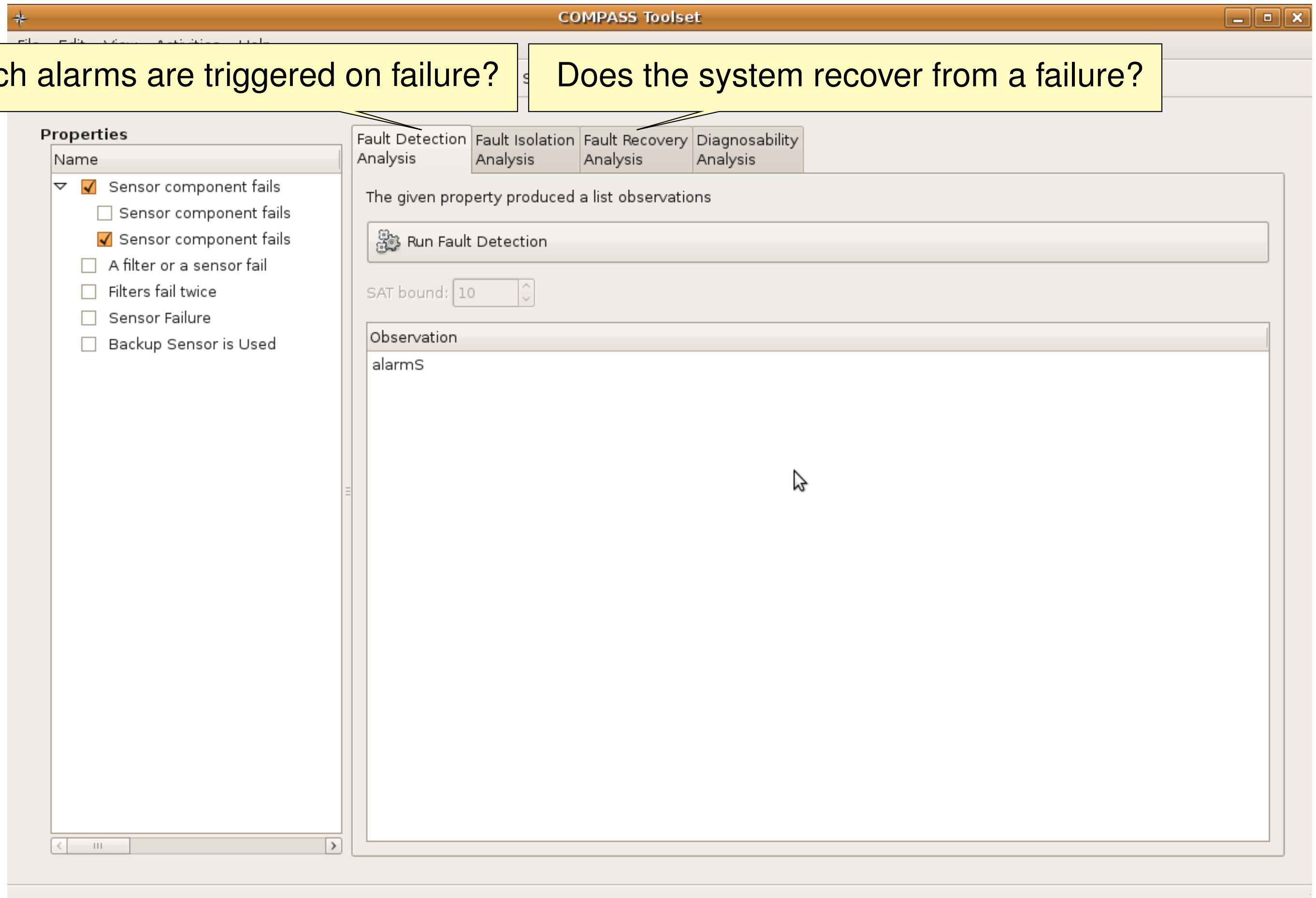
COMPASS Toolset: FDIR Effectiveness



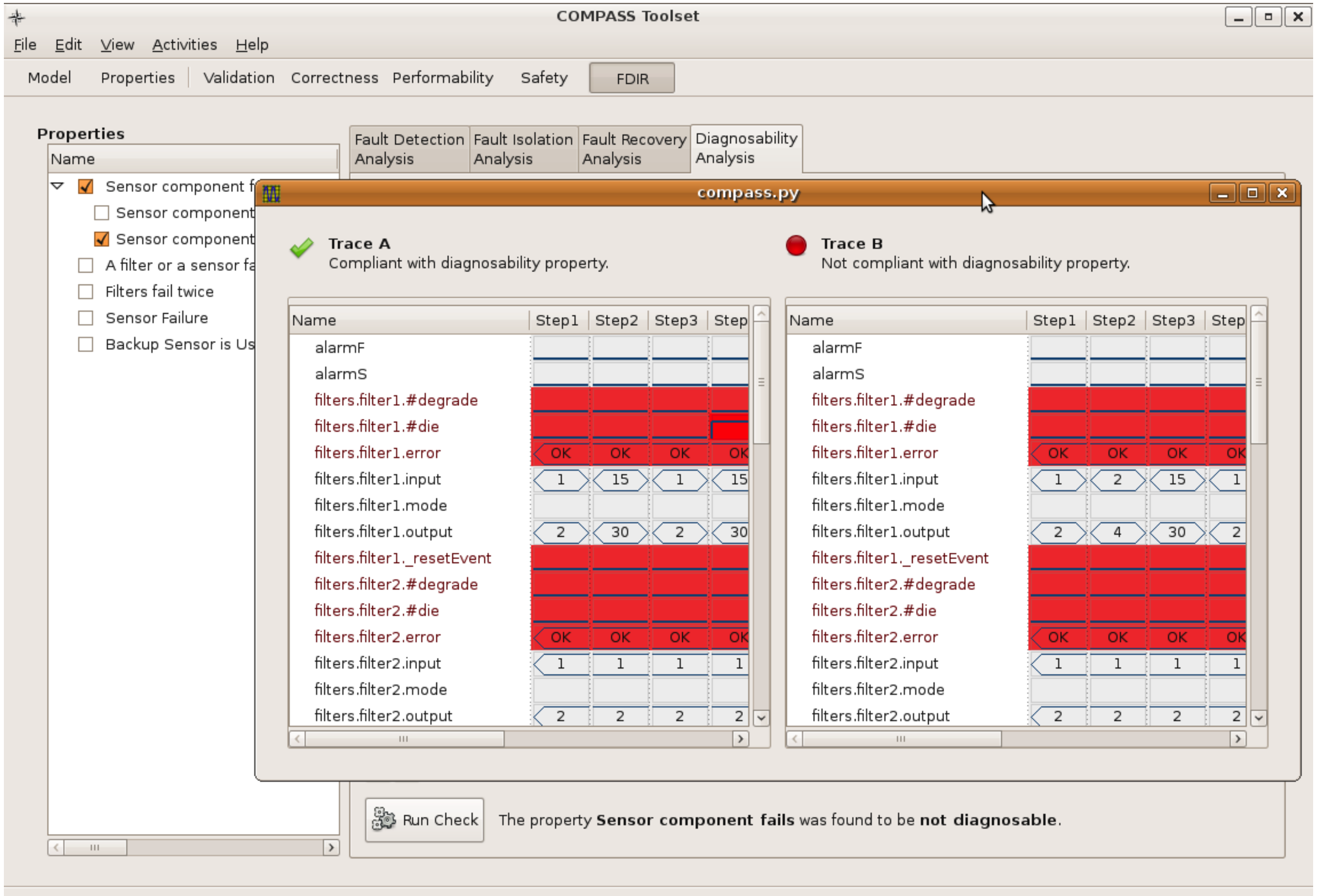
COMPASS Toolset: FDIR Effectiveness

Which alarms are triggered on failure?

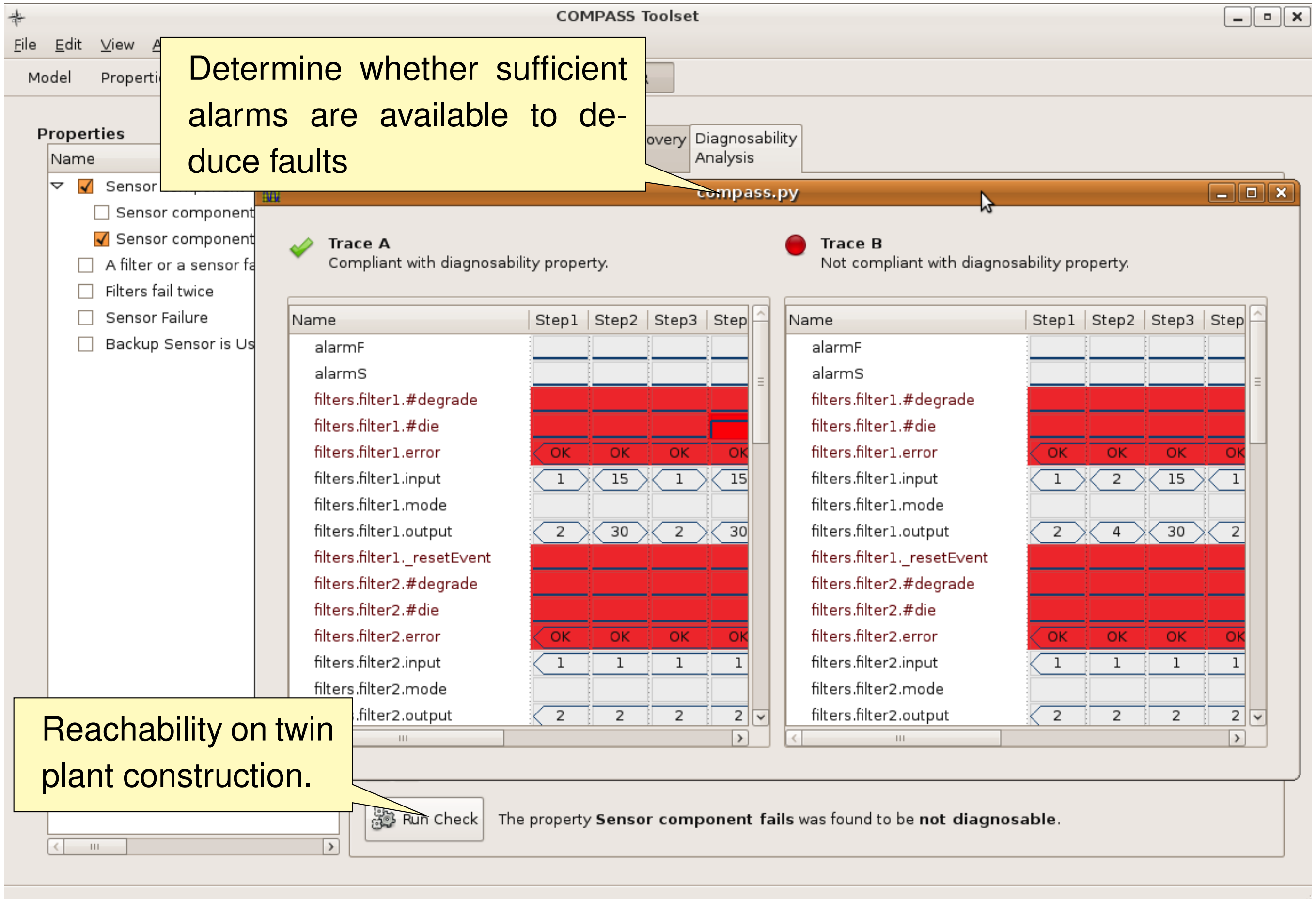
Does the system recover from a failure?



COMPASS Toolset: Diagnosability



COMPASS Toolset: Diagnosability



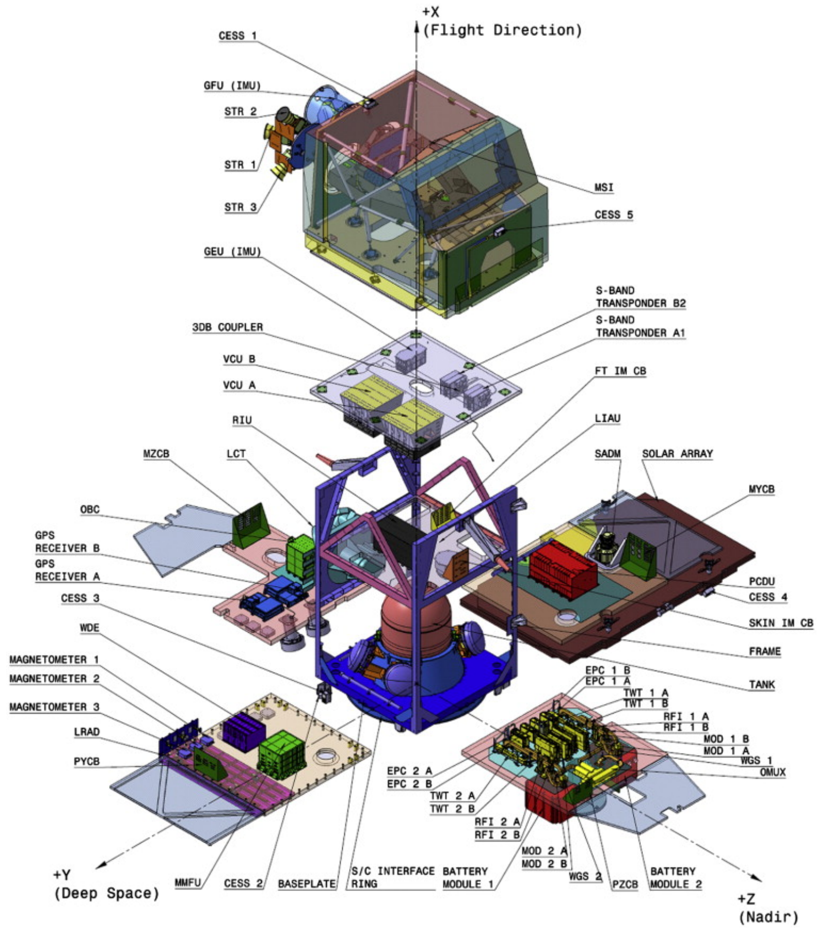
Determine whether sufficient alarms are available to deduce faults

Reachability on twin plant construction.

Outline

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Case Study: Platform of ESA Satellite

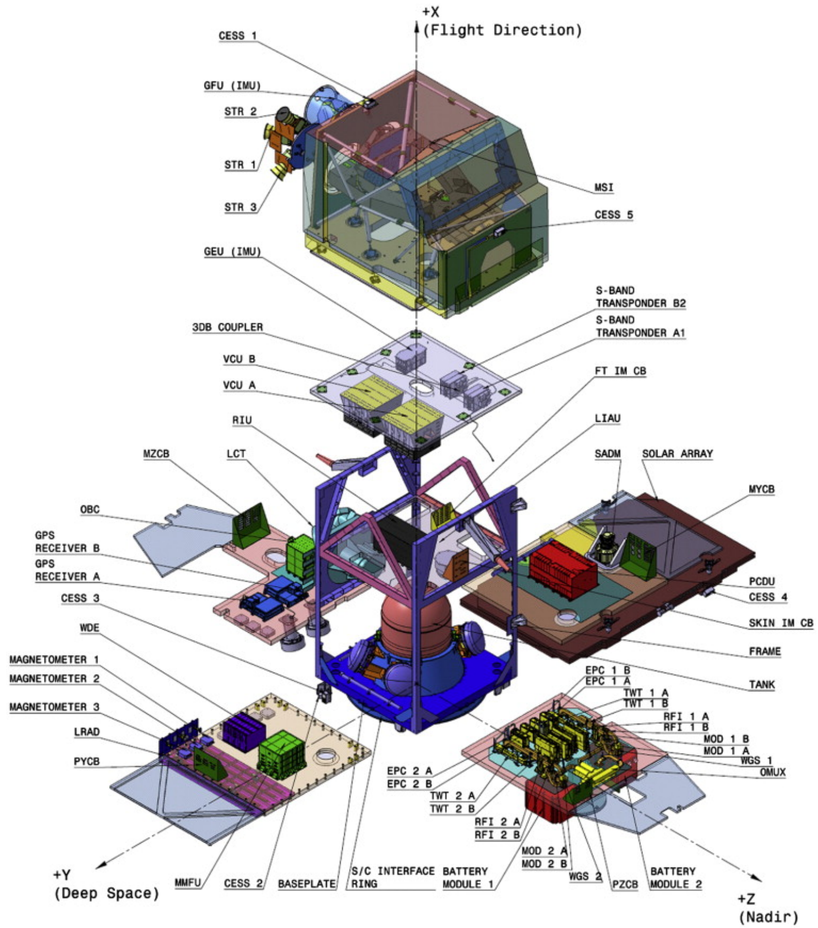


Note: Shown picture is not from the case study

Platform for basic functionality:

- control & data unit
- propulsion
- telemetry, tracking & cmd
- power
- attitude & orbit control
- reconfiguration modules
- etc.

Case Study: Platform of ESA Satellite



Note: Shown picture is not from the case study

Platform for basic functionality:

- control & data unit
- propulsion
- telemetry, tracking & cmd
- power
- attitude & orbit control
- reconfiguration modules
- etc.

FDIR:

- redundancies + recovery
- compensation algorithms
- failure isolation schemes
- omnipresent in satellite

AADL Model of Satellite Platform

Verification & validation objectives

- Ensure correct handling of nominal and degraded conditions by fault management system
- Ensure that performance and risks are within specified limits

AADL Model of Satellite Platform

Verification & validation objectives

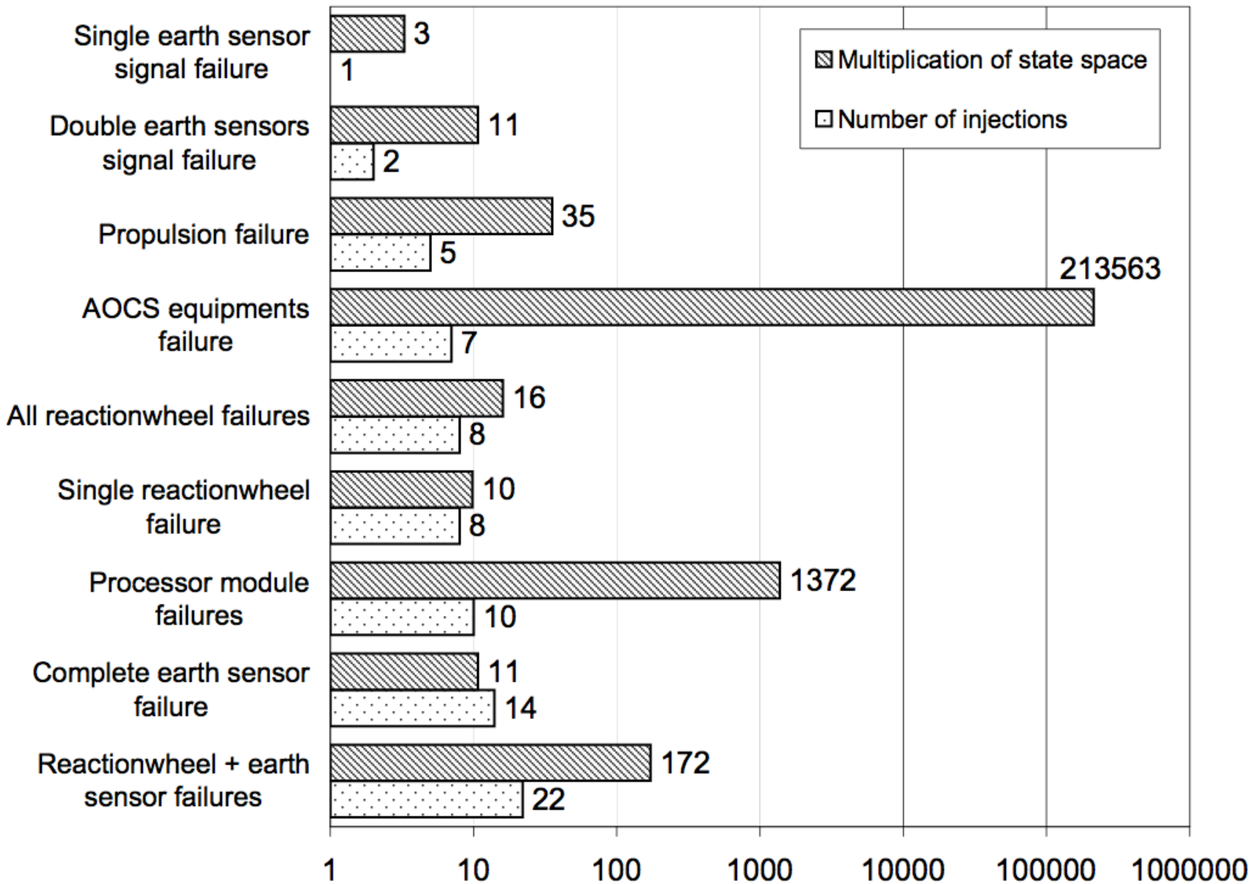
- Ensure correct handling of nominal and degraded conditions by fault management system
- Ensure that performance and risks are within specified limits

Case study characteristics

Model		Requirements	
Metric	Count	Metric	Count
Components	86	Propositional	25
Ports	937	Absence	2
Modes	244	Universality	1
Error models	20	Response	14
Recoveries	16	Probabilistic Invariance	1
Nominal state space	48,421,100	Probabilistic Existence	1
LOC (w/o comments)	3831		

Industrial Evaluation

Effect of Fault Injections



Outline

Conclusion

Epilogue

Achievements

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

Trustworthy aerospace design = AADL modelling + analysis

Conclusion

Epilogue

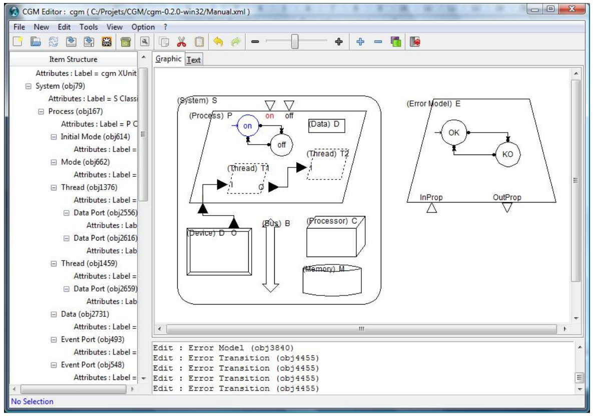
Achievements

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

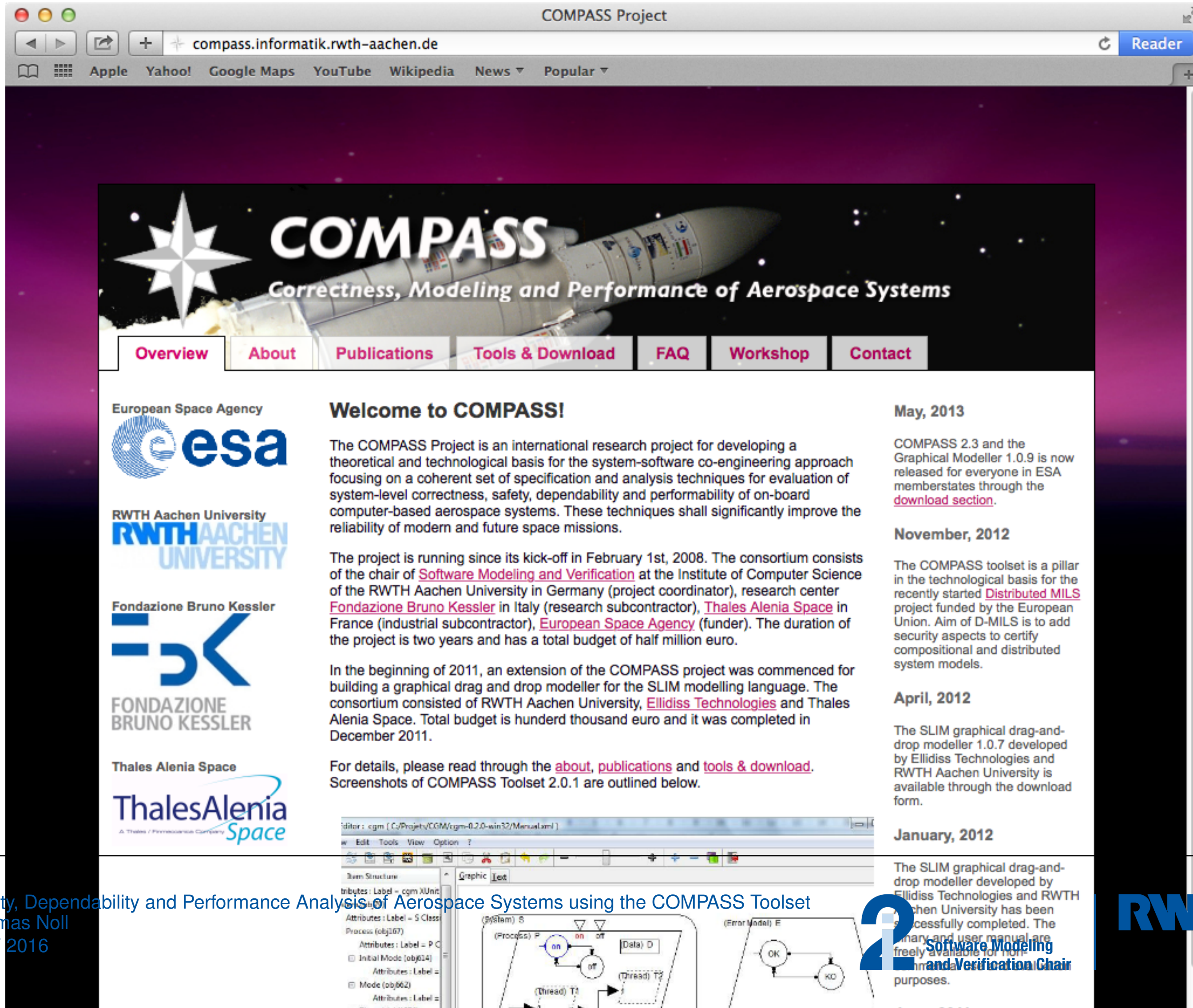
Trustworthy aerospace design = AADL modelling + analysis

Ongoing efforts

- Compositional model checking
- Launcher systems (ESA HASDEL)
- Taxonomy of system and software properties (ESA CATSY)
- Integration of security analysis (EU D-MILS)



The COMPASS Web Site



Conclusion

The End