



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

Overview

Motivation

COMPASS Project Overview

System Specifications

Error Modelling

Analysis Facilities

Industrial Evaluation

Conclusion

Motivation

Outline

Motivation

COMPASS Project Overview

System Specifications

Error Modelling

Analysis Facilities

Industrial Evaluation

Conclusion

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



Motivation

Fault-Tolerant Space System Architectures

Space system requirements

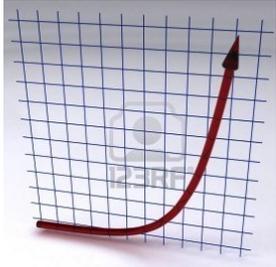
- Must offer **service without interruption** for a very long time – typically years or decades
- **Failures** are costly and may severely damage reputations:
 - Ariane 5 crash in 1996 due to arithmetic overflow
 - Launch failure of Phobos-Grunt sample return mission
- “Five nines” (99.999 %) dependability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

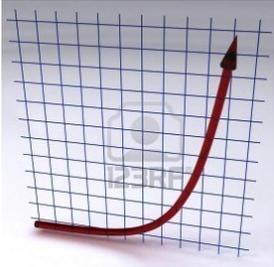
Challenges for Verification & Validation



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**

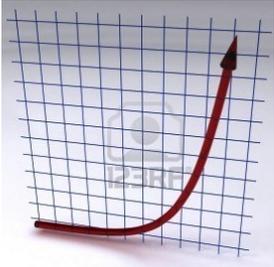
- Apollo (1970) 8,500 LOC
- Space Shuttle (1980) 470,000 LOC
- ISS (1995) 1,000,000 LOC



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

COMPASS Project Overview

Outline

Motivation

COMPASS Project Overview

System Specifications

Error Modelling

Analysis Facilities

Industrial Evaluation

Conclusion

COMPASS Project Overview

COorrectness, M odelling and P erformance 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

COorrectness, M odelling and P erformance 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**
(**S**ystem-**L**evel **I**ntegrated **M**odelling 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 Overview

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

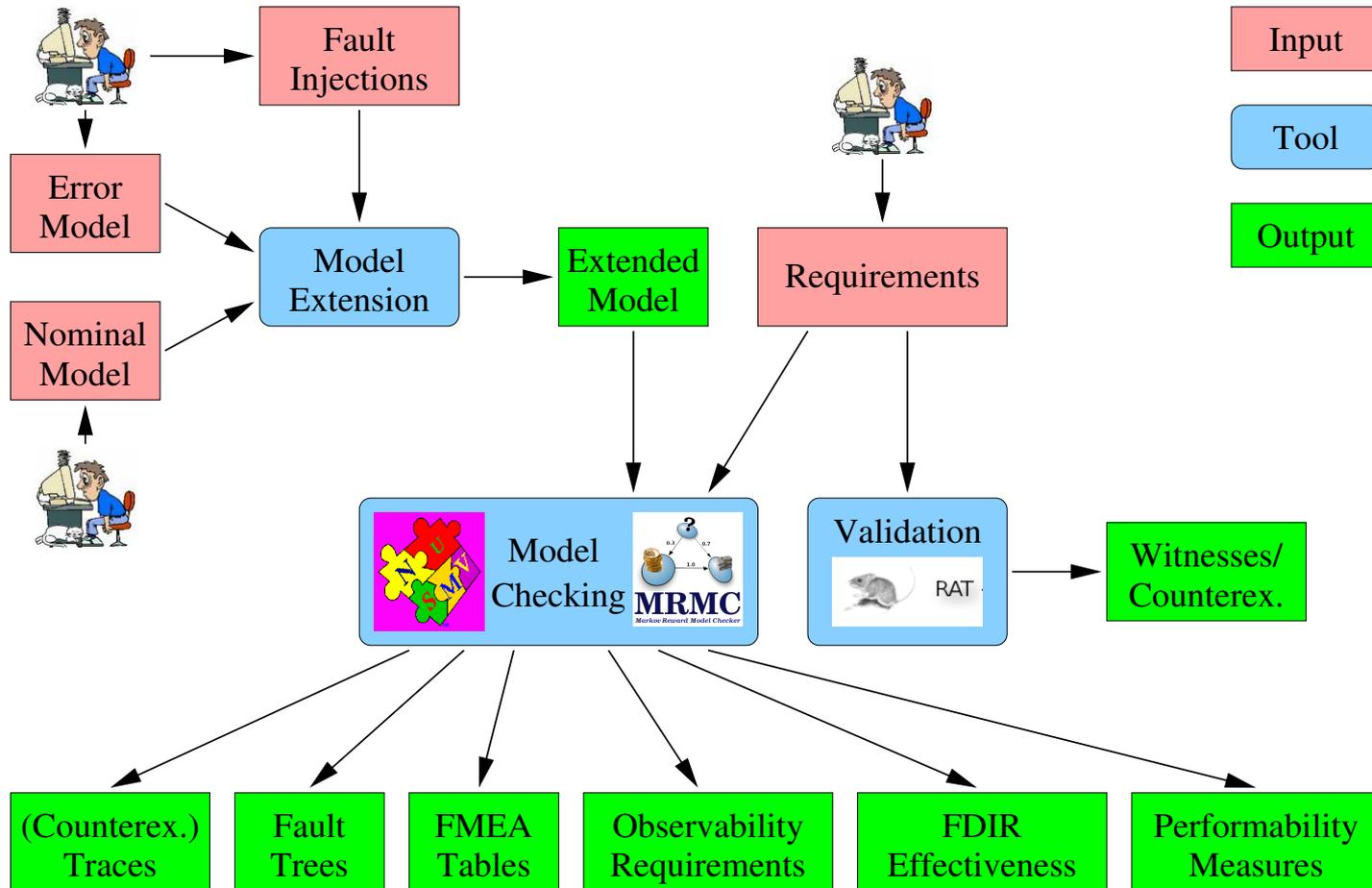
Funding & supervision

- **European Space Agency**



COMPASS Project Overview

COMPASS Methodology



System Specifications

Outline

Motivation

COMPASS Project Overview

System Specifications

Error Modelling

Analysis Facilities

Industrial Evaluation

Conclusion

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

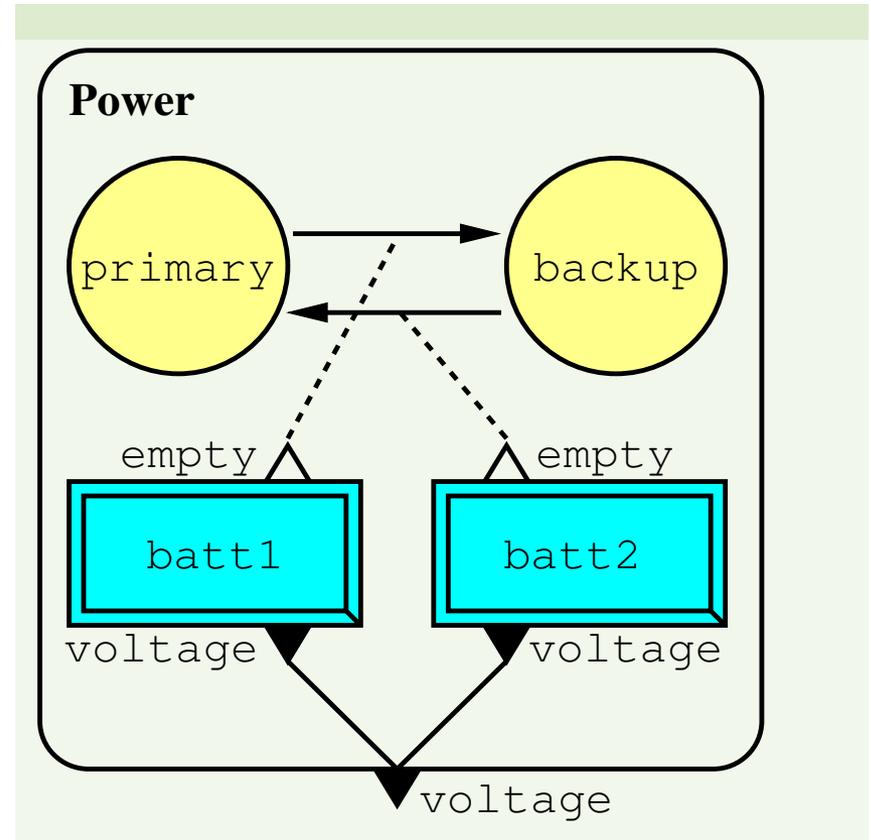


System Specifications

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



System Specifications

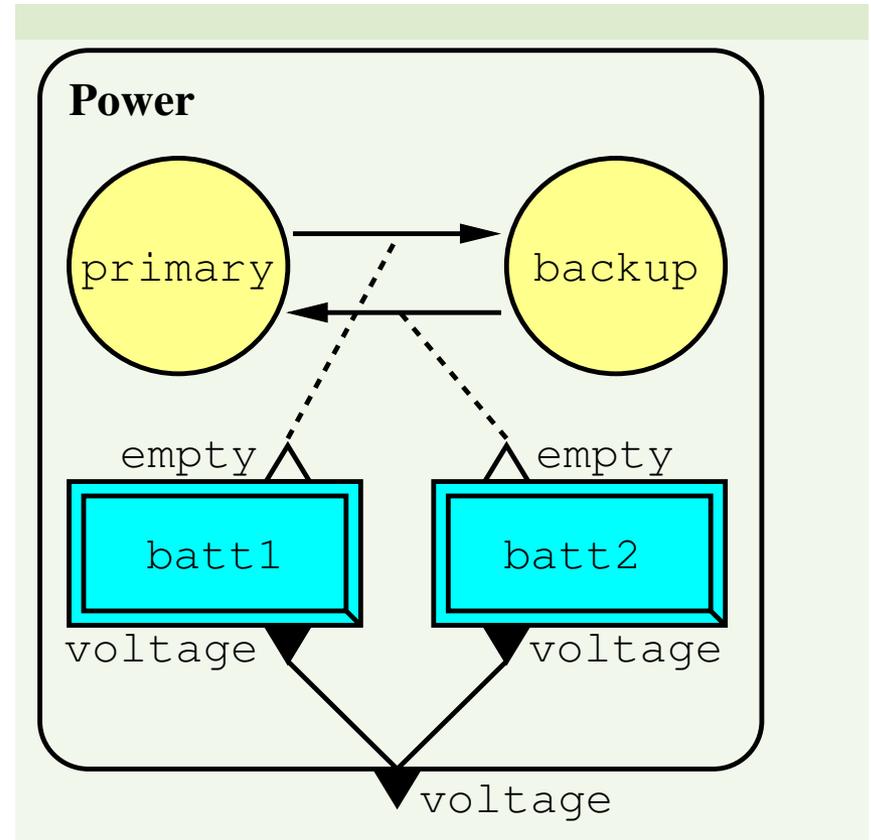
Running Example: Redundant Power System

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**



System Specifications

Modelling a Battery

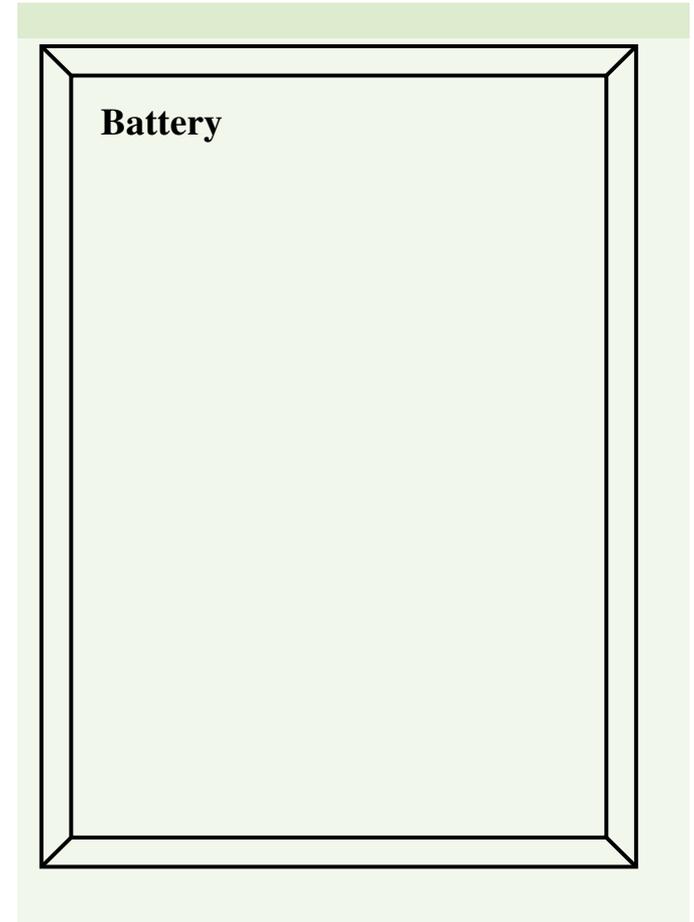
Component **type** and **implementation**:

```
device type Battery
```

```
end Battery;
```

```
device implementation Battery.Imp
```

```
end Battery.Imp;
```



System Specifications

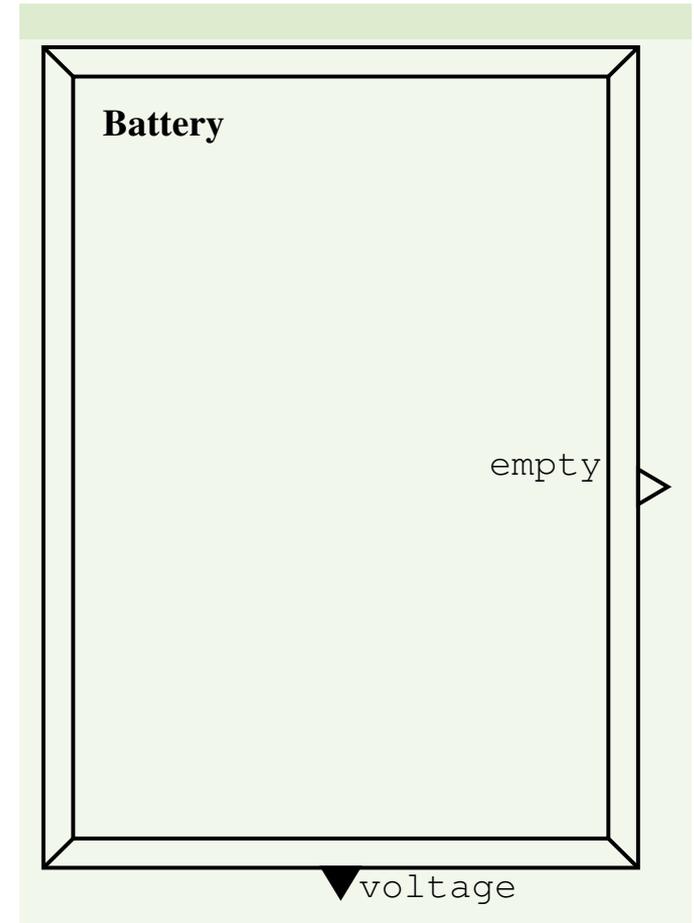
Modelling a Battery

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;
```



System Specifications

Modelling a Battery

Adding **mode behaviour**:

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

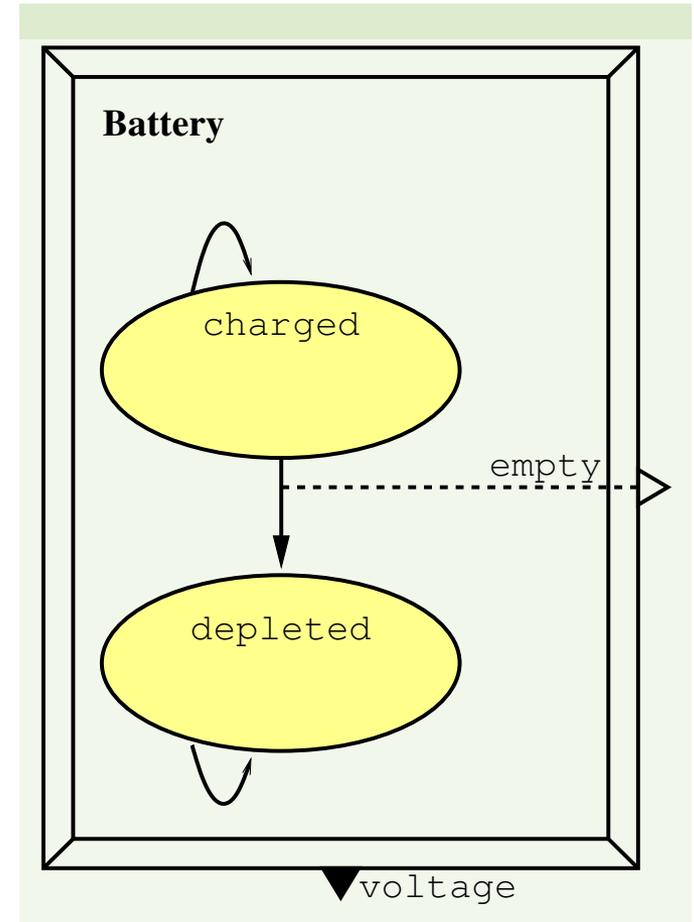
```
device implementation Battery.Imp
```

```
  modes
    charged: initial mode
```

```
    depleted: mode
```

```
  transitions
    charged -[]-> charged;
    charged -[empty]-> depleted;
    depleted -[]-> depleted;
```

```
end Battery.Imp;
```



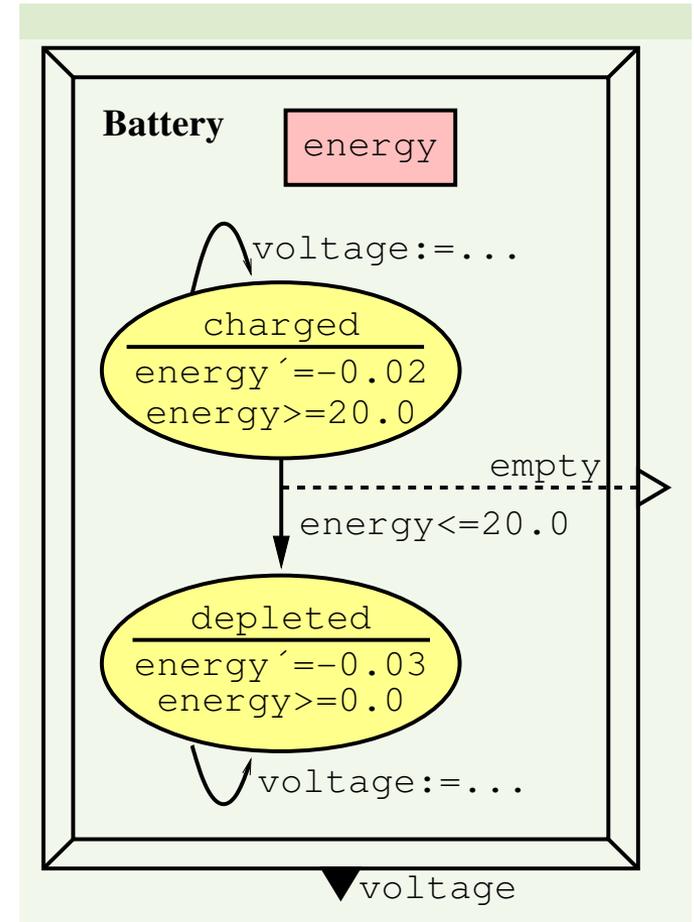
System Specifications

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;
```



System Specifications

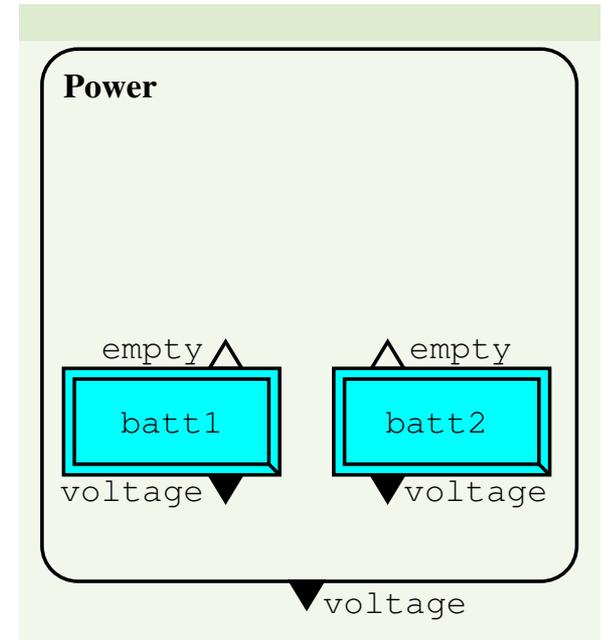
Modelling the Redundant Power System

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;
```



System Specifications

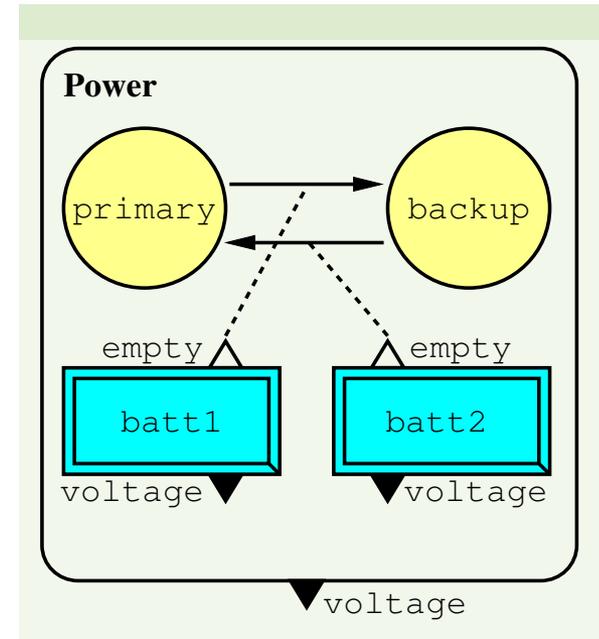
Modelling the Redundant Power System

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;
```



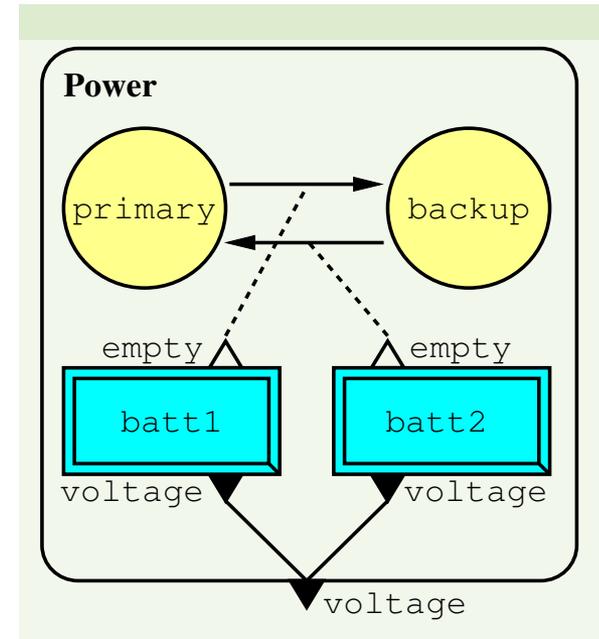
System Specifications

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;
```



Error Modelling

Outline

Motivation

COMPASS Project Overview

System Specifications

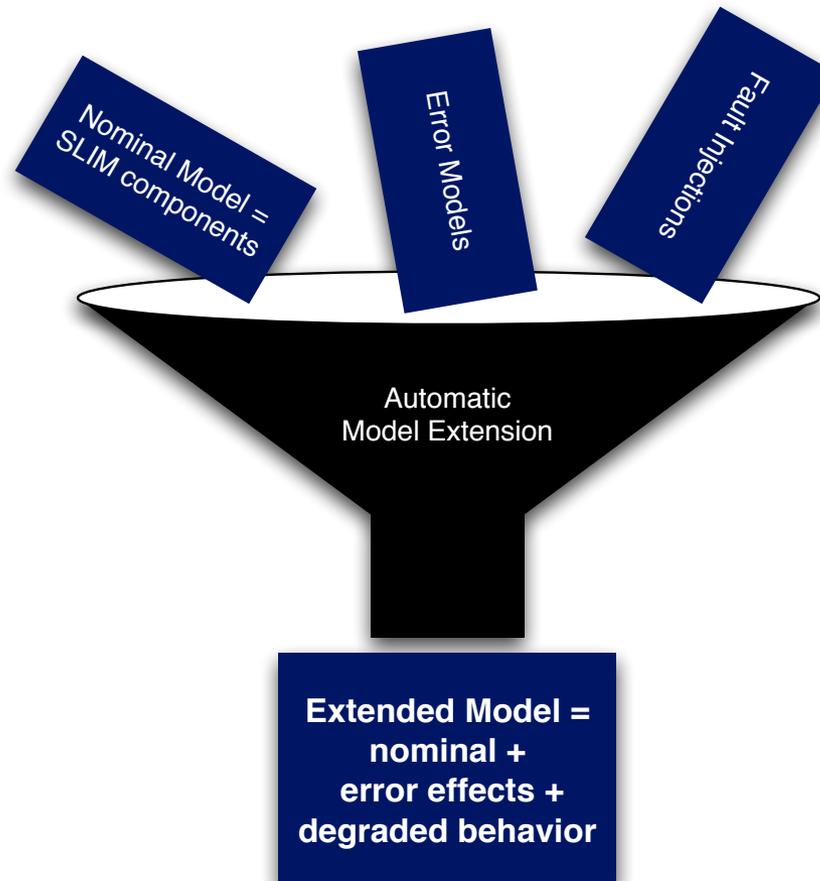
Error Modelling

Analysis Facilities

Industrial Evaluation

Conclusion

Integrating Erroneous and Nominal Behaviour



Error Modelling

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 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 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;
```

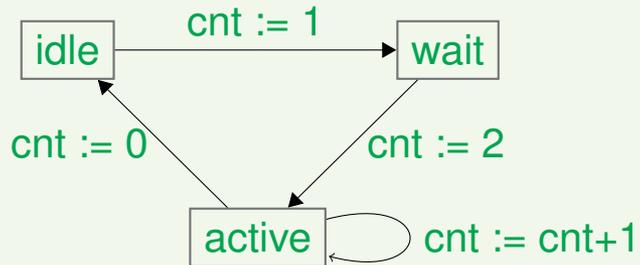
Fault injection

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

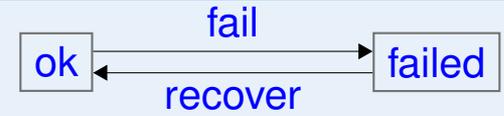
Error Modelling

Model Extension by Example

Nominal behaviour



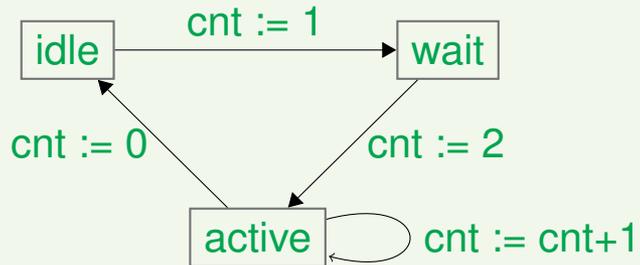
Error behaviour



Error Modelling

Model Extension by Example

Nominal behaviour



Fault injection

failed: $cnt := -1$

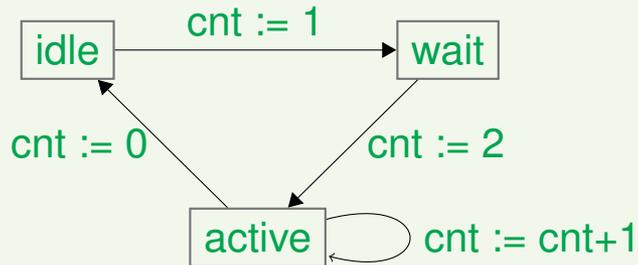
Error behaviour



Error Modelling

Model Extension by Example

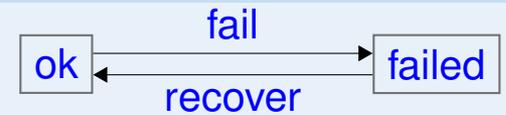
Nominal behaviour



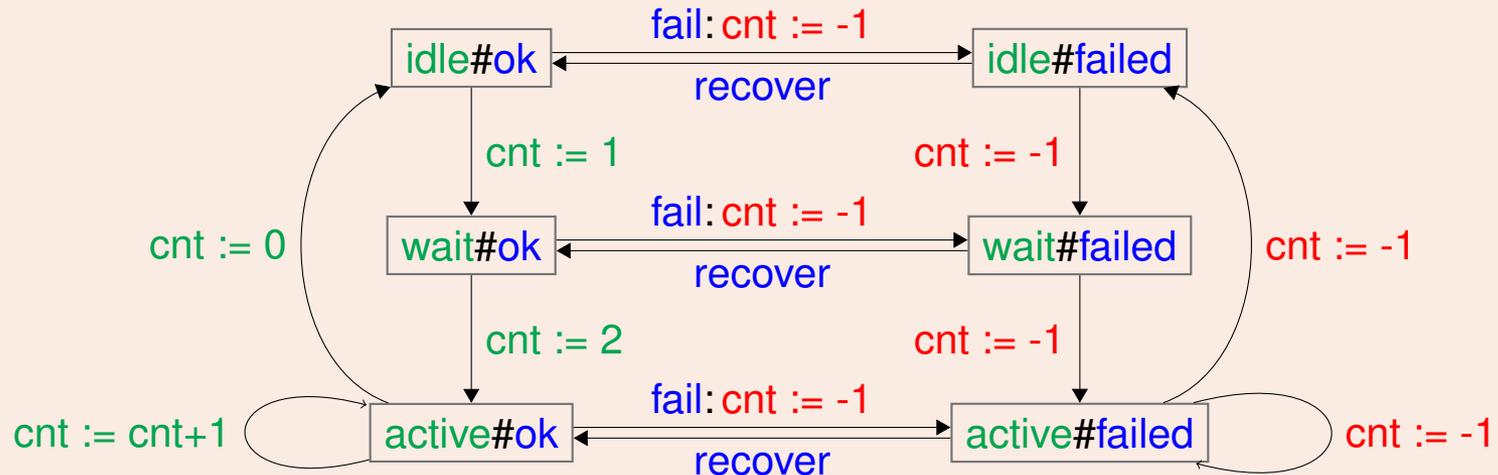
Fault injection

failed: cnt := -1

Error behaviour



Automatically extended model



Analysis Facilities

Outline

Motivation

COMPASS Project Overview

System Specifications

Error Modelling

Analysis Facilities

Industrial Evaluation

Conclusion

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 continuously within time bound $[t_1, t_2]$.

Requirements: Patterns, not Formulas!

Instantiated patterns

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

Requirements: Patterns, not Formulas!

Instantiated patterns

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

↓
(by automatic transformation)



Logic

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

Requirements: Patterns, not Formulas!

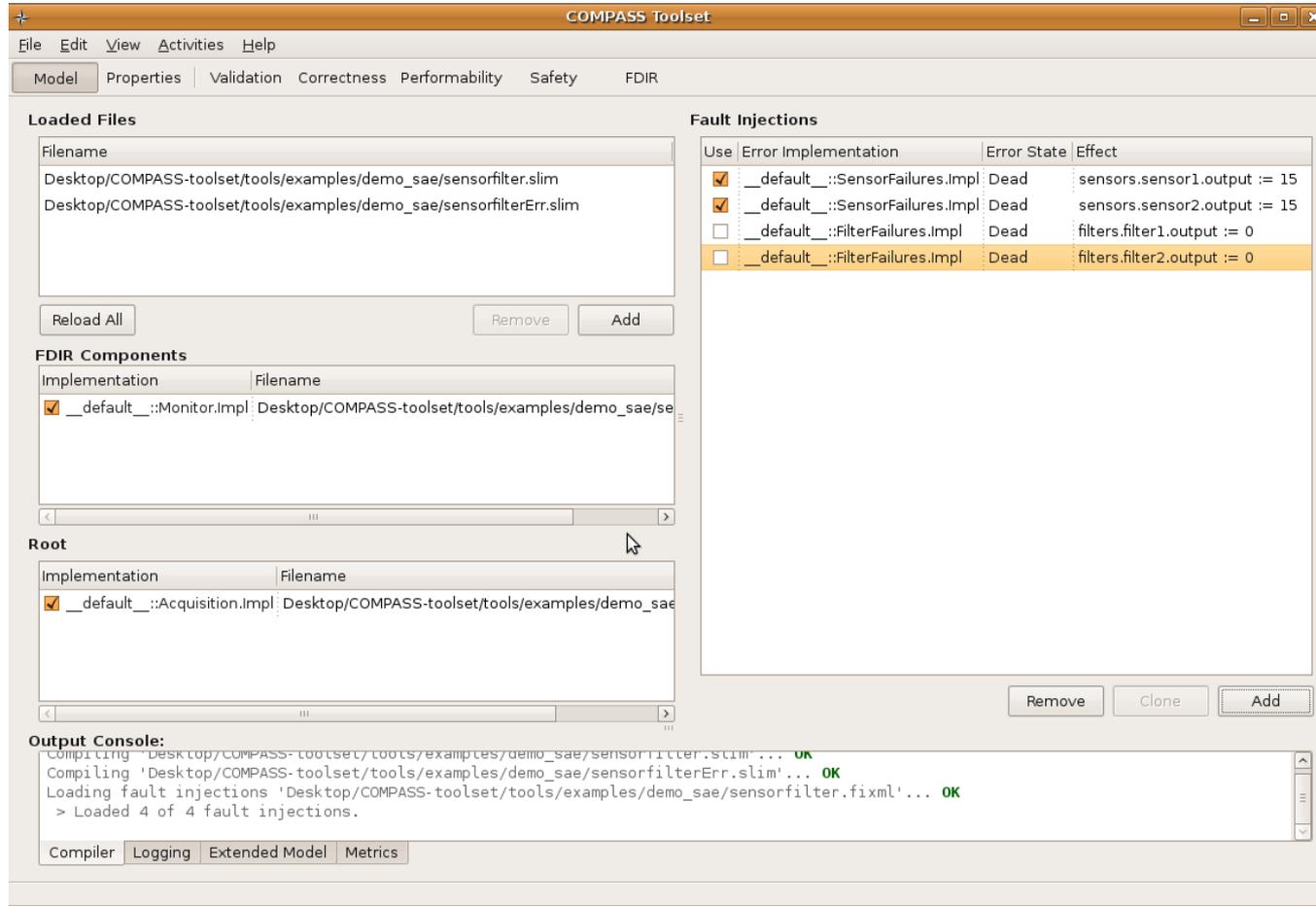
Logic

- $\square(80 \leq \text{voltage} \wedge \text{voltage} \leq 90)$ (Linear Temporal Logic, LTL)
- $\mathcal{P}_{>0.98}(\square^{[0,10]}(\text{voltage} \geq 80))$ (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

The screenshot shows the COMPASS Toolset main window. The title bar reads "COMPASS Toolset". The menu bar includes "File", "Edit", "View", "Activities", and "Help". The main area is divided into several sections:

- Loaded Files:** A table with columns "Filename" and "Implementation". It lists two files: "Desktop/COMPASS-toolset/tools/examples/demo_sae/sensorfilter.slim" and "Desktop/COMPASS-toolset/tools/examples/demo_sae/sensorfilterErr.slim". Below the table are "Reload All", "Remove", and "Add" buttons.
- FDIR Components:** A table with columns "Implementation" and "Filename". It lists one component: "__default__::Monitor.impl" with filename "Desktop/COMPASS-toolset/tools/examples/demo_sae/se...".
- Root:** A table with columns "Implementation" and "Filename". It lists one component: "__default__::Acquisition.impl" with filename "Desktop/COMPASS-toolset/tools/examples/demo_sae...".
- Fault Injections:** A table with columns "Use", "Error Implementation", "Error State", and "Effect". It lists four entries, all with "Use" checked and "Error State" set to "Dead". The "Effect" column contains: "sensors.sensor1.output := 15", "sensors.sensor2.output := 15", "filters.filter1.output := 0", and "filters.filter2.output := 0".
- Output Console:** A text area showing the following output:

```
Compiling 'Desktop/COMPASS-toolset/tools/examples/demo_sae/sensorfilterErr.slim'... OK
Compiling 'Desktop/COMPASS-toolset/tools/examples/demo_sae/sensorfilter.slim'... OK
Loading fault injections 'Desktop/COMPASS-toolset/tools/examples/demo_sae/sensorfilter.fixml'... OK
> Loaded 4 of 4 fault injections.
```

Annotations in yellow boxes point to specific features:

- "Adding requirements" points to the "Loaded Files" section.
- "Loading SLIM models" points to the "Loaded Files" section.
- "Adding fault injections" points to the "Fault Injections" table.

COMPASS Toolset: Simulation

COMPASS Toolset

File Edit View Activities Help

Model Properties Validation Correctness Performability Safety FDIR

Properties

Name

- Sensor component fails
- A filter or a sensor fail
- Filters fail twice
- Sensor Failure
- Backup Sensor is Used

Simulation

Model Simulation Deadlock Checking Model Checking

Model extended by fault injections

Guided by Transitions Run Length: 10 Restart Jump

Name	Impl	Step1	Step2	Step3	Name
mode					monitor.alarmF
					monitor.alarmS
		OK	OK	OK	monitor.mode
					monitor.switchF
					monitor.switchS
		2	2	30	monitor.valueF
		1	1	15	monitor.valueS
					mybus.mode
		Primary	Primary	Primary	sensors.mode
		1	1	15	sensors.output
		OK	Drifted	Dead	sensors.sensor1.#die
					sensors.sensor1.#dieByDrift
					sensors.sensor1.#drift
					sensors.sensor1.error

Transitions

- OK [_resetEvent] -> OK;
- Drifted [_resetEvent] -> Drifted;
- Dead [_resetEvent] -> Dead;
- Drifted [#dieByDrift then _errorState := ...]
- OK [#drift then _errorState := _Drifted] -> ...
- OK [#die then _errorState := _Dead] -> ...

Name: [] Stored: No Filter Edit

COMPASS Toolset: Simulation

Simulation states

Name	Impl	Step1	Step2	Step3	Name
mode					monitor.alarmF
					monitor.alarmS
					monitor.mode
			OK	OK	monitor.switchF
					monitor.switchS
					monitor.valueF
					monitor.valueS
					mybus.mode
					sensors.mode
					sensors.output
					sensors.sensor1.#die
					sensors.sensor1.#dieByDrift
					sensors.sensor1.#drift
					sensors.sensor1.error

Choice of transition

Transitions

- OK [_resetEvent] -> OK;
- Drifted [_resetEvent] -> Drifted;
- Dead [_resetEvent] -> Dead;
- Drifted [#dieByDrift then _errorState := ...]
- OK [#drift then _errorState := _Drifted] -> ...
- OK [#die then _errorState := _Dead] -> ...

COMPASS Toolset: Model Checking

COMPASS Toolset

File Edit View Activities Help

Model Properties Validation Correctness Performability Safety FDIR

Properties

Name MC

- Monitor reacts to filter failures ✓
- Sensor component fails
- A filter or a sensor fail ●
- Filters fail twice ●
- Sensor Failure
- Backup Sensor is Used

Model Simulation Deadlock Checking Model Checking

Run Model Checking Model extended by Fault Injections

Model Checker Options:

● The property is false

The CTL property: `value = 0 and filters.mode = mode:Backup` has been found false. A counter-example is shown

Name	Step1
alarmF	
alarmS	
filters.filter1.input	1
filters.filter1.mode	
filters.filter1.output	2
filters.filter2.input	1
filters.filter2.mode	
filters.filter2.output	2
filters.input	1
filters.mode	Primary
filters.output	2
mode	
monitor.alarmF	

Name: ✓ Stored: No Filter Edit

COMPASS Toolset: Model Checking

The screenshot displays the COMPASS Toolset interface with several callouts explaining key features:

- SMT model checking of LTL formula for hybrid case:** Points to the 'Model Checking' button in the top toolbar.
- BDD model checking of CTL formula for discrete case:** Points to the 'Model Checking' button in the top toolbar.
- Choice of property to be verified:** Points to the list of properties on the left, where 'Filters fail twice' is selected.
- Counterexample when property does not hold:** Points to the 'Step1' trace window showing a sequence of states where the property fails.

The main window shows the 'Run Model Checking' dialog with the following configuration:

- Model: `Model extended by Fault injections`
- Property: `value = 0 and filters.mode = mode:Backup`
- Result: `false`
- Message: `The CTL property: value = 0 and filters.mode = mode:Backup has been found false. A counter-example is shown`

The counterexample trace (Step1) shows the following sequence of states:

Variable	Value
filters.filter2.mode	1
filters.filter2.output	2
filters.input	1
filters.mode	Primary
filters.output	2

COMPASS Toolset: Fault Tree Analysis

The screenshot displays the COMPASS Toolset interface. The main window has a menu bar (File, Edit, View, Activities, Help) and a toolbar with tabs for Model, Properties, Validation, Correctness, Performability, Safety, and FDIR. The Properties panel on the left shows a list of components under the 'FT' (Fault Tree) category, with 'Sensor component fails' checked. The main workspace contains a 'Generate Fault Tree' button and 'Model Checker Options' including 'Use BDD' and 'Dynamic'. A separate window titled 'FSAP: Fault Tree Displayer' shows a hierarchical fault tree diagram. The diagram's root node is 'The Event Occurs', which branches into four intermediate nodes labeled 'gate_1', 'gate_2', 'gate_3', and 'gate_4'. Each gate node further decomposes into basic events, represented by circles labeled 'E1', 'E2', 'E3', and 'E4'.

COMPASS Toolset: Fault Tree Analysis

The image shows the COMPASS Toolset interface with the 'Safety' tab selected. The 'Properties' panel on the left shows 'Sensor component fails' checked. The 'Fault Tree' section has 'Generate Fault Tree' and 'Model Checker Options' (Use BDD, Dynamic) visible. A yellow callout box points to the 'Generate Fault Tree' button with the text: "Relate faults to higher-level failures, e.g., 'which faults lead to undervoltage?'".

Below the main interface is a window titled 'FSAP: Fault Tree Displayer'. It shows a fault tree diagram with a root event 'The Load is open' and several intermediate events (gates) leading to basic events (sensors). A yellow callout box points to the top of the tree with the text: "Fault tree contains (priority) AND/OR-gates. Edges represent triggers".

COMPASS Toolset: Probabilistic Risk Assessment

The screenshot displays the COMPASS Toolset interface. The main window has a menu bar (File, Edit, View, Activities, Help) and a toolbar with tabs for Model, Properties, Validation, Correctness, Performability, Safety, and FDIR. The Properties panel on the left lists several checked options: Sensor component fails, A filter or a sensor fail, Filters fail twice, Sensor Failure, and Backup Sensor is Used. The main workspace shows the 'Generate FMEA Table' button, with 'Cardinality' set to 1 and 'SAT bound' set to 10. The 'Compact FMEA' checkbox is checked. An inset window titled 'FSAP: Fault Tree Displayer' shows a fault tree diagram with a top event 'Top_Level_Event' (P = 1.111600e-03, C = 5.558000e-03) and a basic event 'sensors.sensor1.output >= 15'. The right side of the main window displays the resulting FMEA table with columns for event name, description, and occurrence rate.

COMPASS Toolset: Probabilistic Risk Assessment

COMPASS Toolset

File Edit View Activities Help

Model Properties Validation Correctness Performability Safety

Properties

- ✓ Sensor component fails
- ✓ Sensor component fails
- ✓ Sensor component fails
- ✓ A filter or a sensor fail
- ✓ Filters fail twice
- ✓ Sensor Failure
- ✓ Backup Sensor is Used

Fault Tree Generation

Failure Mode Effect Analysis

Fault Tree Evaluation

The resulting FMEA table is presented

Generate FMEA Table

Cardinality: 1 SAT bound: 10

Dynamic FMEA Compact FMEA

FSAP: Fault Tree Displayer

File Actions View

Contents File: /home/compasseval/Desktop/COMPASS

Resources File: /home/compasseval/Desktop/COMPASS

To

sensors.sensor1.output >= 15

Top_Level Event
P = 1.111600e-03
C = 5.558000e-03

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

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

COMPASS Toolset: FMEA Analysis

The screenshot displays the COMPASS Toolset interface with the 'Safety' tab selected. The 'Properties' panel on the left lists several checked items: 'Sensor component fails', 'A filter or a sensor fail', 'Filters fail twice', 'Sensor Failure', and 'Backup Sensor is Used'. The main workspace shows the 'Failure Mode Effect Analysis' tab. A 'Generate FMEA Table' button is visible, along with 'Cardinality: 1' and 'SAT bound: 10' settings. The 'Compact FMEA' checkbox is checked. Below these settings, a table displays the resulting FMEA table with 5 rows of data.

Num	ID	Failure Model	Failure Effect
1	1-1	sensors.sensor1_errorSubcomponent.#die = True	root.sc_sensors.sc_sensor1.data_output >= 0sd7_15
2	1-3	sensors.sensor1_errorSubcomponent.#die = True	(root.data_value >= 0sd7_15 root.data_value = 0sd7_0)
3	1-5	sensors.sensor1_errorSubcomponent.#die = True	root.data_value >= 0sd7_15
4	1-6	sensors.sensor1_errorSubcomponent.#die = True	root.sc_sensors.mode = mode_Backup
5	7-3	filters.filter1_errorSubcomponent.#die = True	(root.data_value >= 0sd7_15 root.data_value = 0sd7_0)

COMPASS Toolset: FMEA Analysis

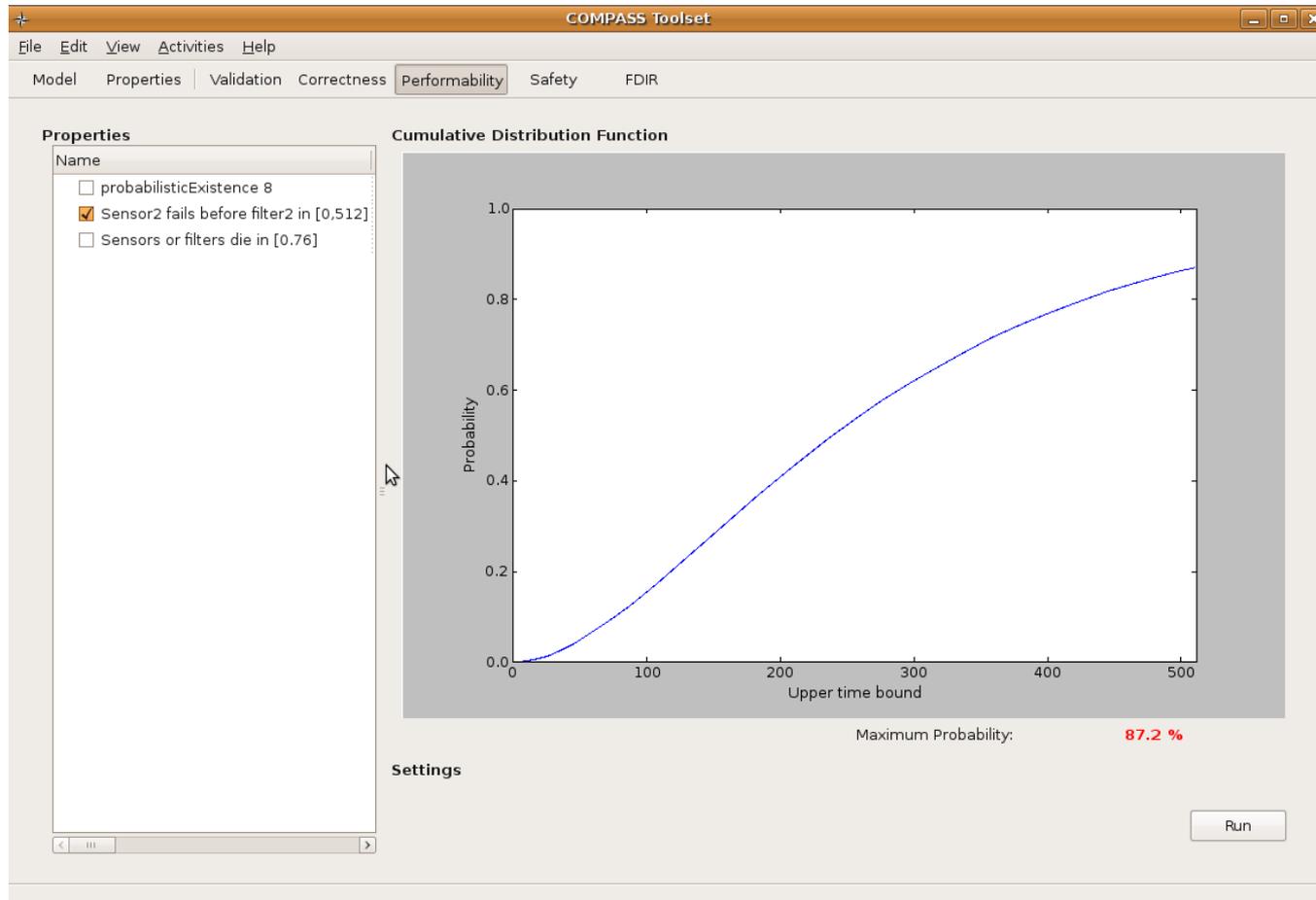
The screenshot shows the COMPASS Toolset interface. The main workspace displays the following table of failure effects:

		Failure Effect
1	1-3 sensors.sensor2_errorSubcomponent.#die = True	root.sc_sensors
2	1-3 sensors.sensor2_errorSubcomponent.#die = True	(root.data_value >= 0sd7_15 root.data_value = 0sd7_0)
3	1-5 sensors.sensor1_errorSubcomponent.#die = True	root.data_value >= 0sd7_15
4	1-6 sensors.sensor1_errorSubcomponent.#die = True	root.sc_sensors.mode = mode_Backup
5	7-3 filters.filter1_errorSubcomponent.#die = True	(root.data_value >= 0sd7_15 root.data_value = 0sd7_0)

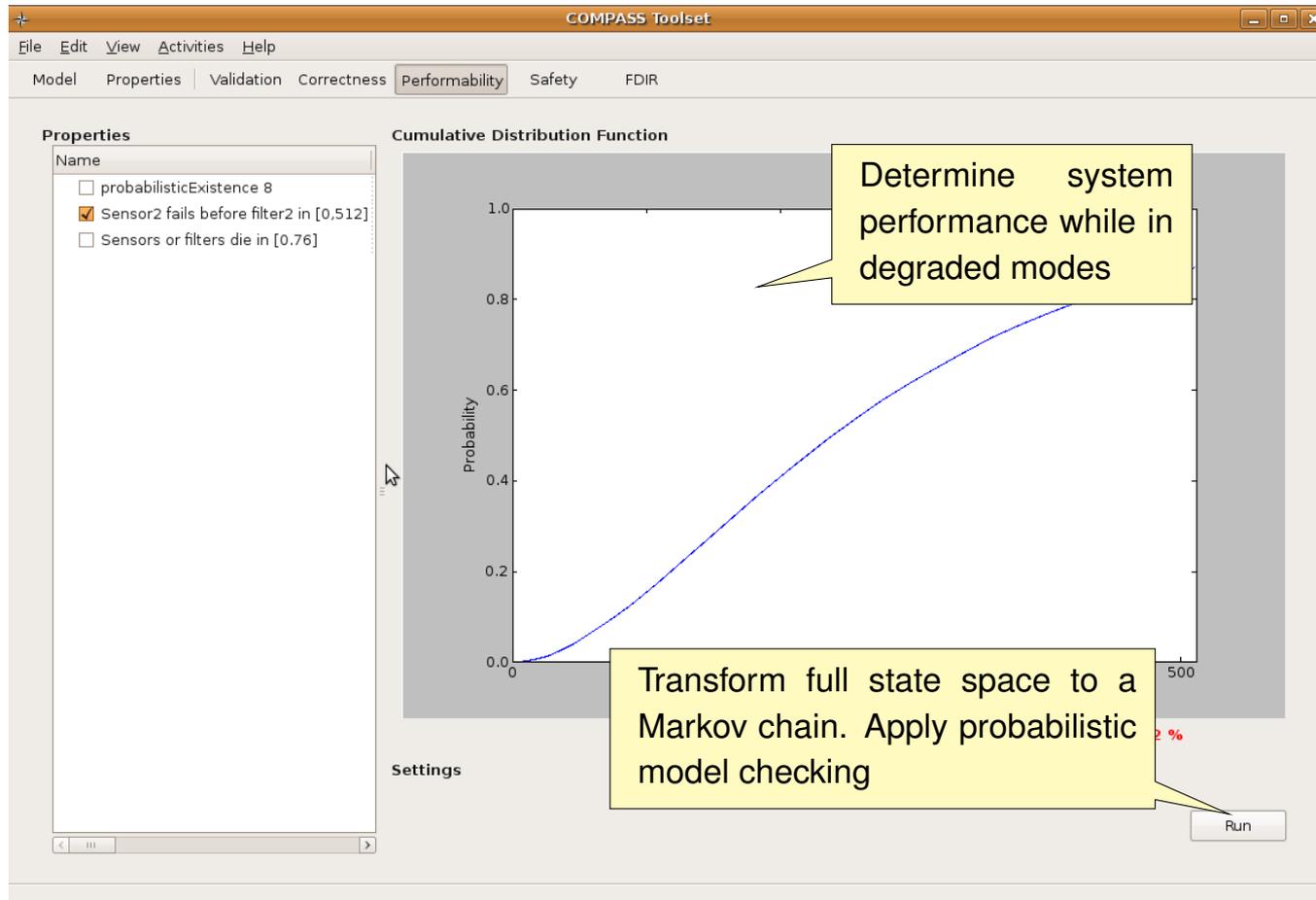
Callouts in the image provide context for the table:

- A yellow callout box above the 'Generate FMEA Table' button contains the text: "Determine effects on nominal model upon combinations of failures".
- A yellow callout box pointing to the first row of the table contains the text: "What happens when sensor dies?".
- A yellow callout box pointing to the second row of the table contains the text: "Value out of range!".

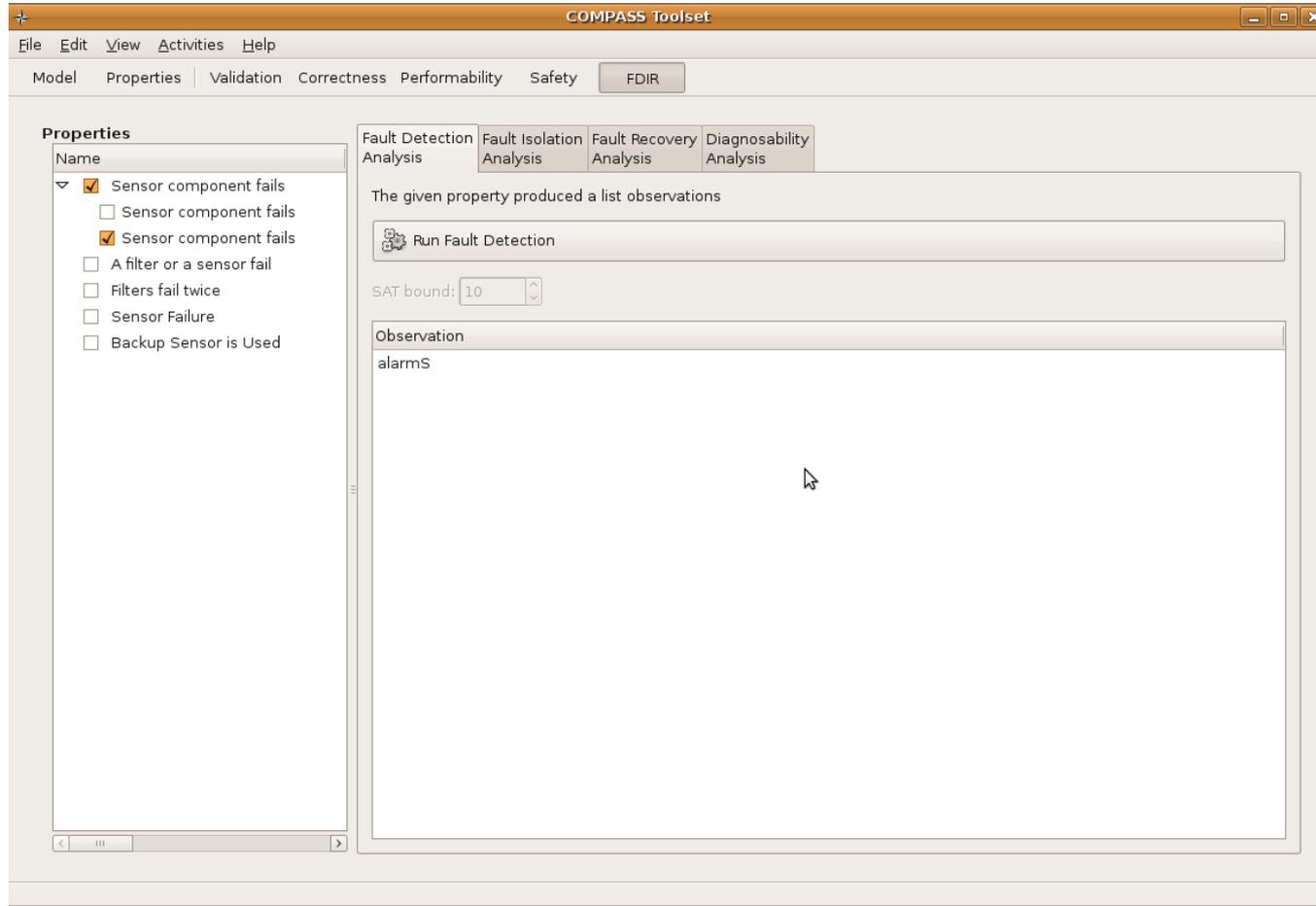
COMPASS Toolset: Performability



COMPASS Toolset: Performability



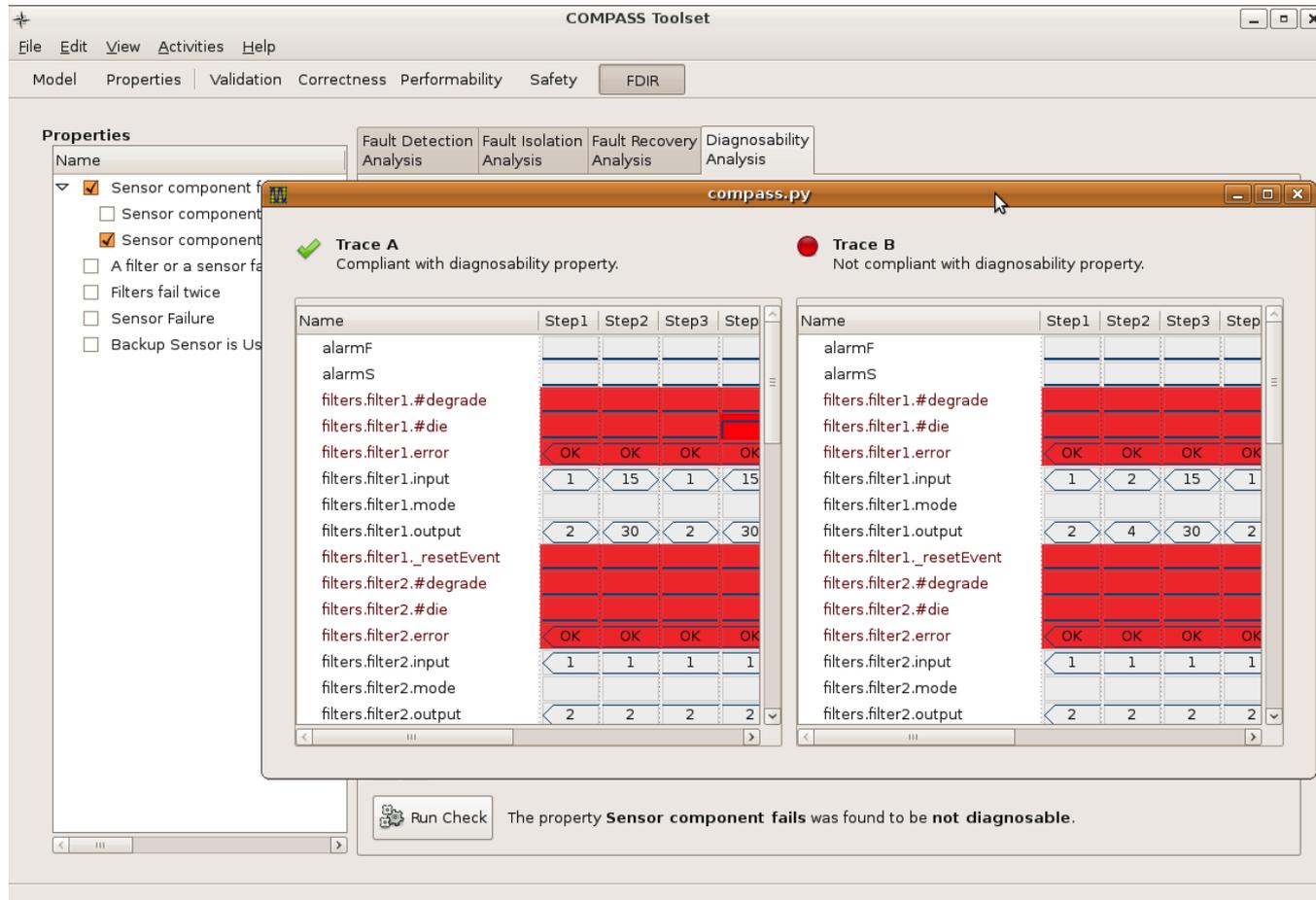
COMPASS Toolset: FDIR Effectiveness



COMPASS Toolset: FDIR Effectiveness

The screenshot displays the COMPASS Toolset interface. On the left, a 'Properties' panel lists several analysis options, with 'Sensor component fails' selected. On the right, the 'Fault Detection Analysis' tab is active, showing a 'Run Fault Detection' button and a 'SAT bound' of 10. Below this, an 'Observation' window displays the text 'alarms'. Two yellow callout boxes are overlaid on the interface: one pointing to the 'Sensor component fails' option with the text 'Which alarms are triggered on failure?', and another pointing to the 'Run Fault Detection' button with the text 'Does the system recover from a failure?'.

COMPASS Toolset: Diagnosability



COMPASS Toolset

File Edit View Activities Help

Model Properties Validation Correctness Performability Safety FDIR

Properties

Name

- Sensor component fails
- Sensor component fails twice
- Sensor component fails and a sensor fails
- A filter or a sensor fails
- Filters fail twice
- Sensor Failure
- Backup Sensor is Used

Trace A
Compliant with diagnosability property.

Name	Step1	Step2	Step3	Step4
alarmF				
alarmS				
filters.filter1.#degrade				
filters.filter1.#die				
filters.filter1.error	OK	OK	OK	OK
filters.filter1.input	1	15	1	15
filters.filter1.mode				
filters.filter1.output	2	30	2	30
filters.filter1._resetEvent				
filters.filter2.#degrade				
filters.filter2.#die				
filters.filter2.error	OK	OK	OK	OK
filters.filter2.input	1	1	1	1
filters.filter2.mode				
filters.filter2.output	2	2	2	2

Trace B
Not compliant with diagnosability property.

Name	Step1	Step2	Step3	Step4
alarmF				
alarmS				
filters.filter1.#degrade				
filters.filter1.#die				
filters.filter1.error	OK	OK	OK	OK
filters.filter1.input	1	2	15	1
filters.filter1.mode				
filters.filter1.output	2	4	30	2
filters.filter1._resetEvent				
filters.filter2.#degrade				
filters.filter2.#die				
filters.filter2.error	OK	OK	OK	OK
filters.filter2.input	1	1	1	1
filters.filter2.mode				
filters.filter2.output	2	2	2	2

Run Check The property **Sensor component fails** was found to be **not diagnosable**.

COMPASS Toolset: Diagnosability

Determine whether sufficient alarms are available to deduce faults

Reachability on twin plant construction.

COMPASS Toolset

File Edit View A

Model Property

Properties

Name

- Sensor
- Sensor component
- Sensor component
- A filter or a sensor fa
- Filters fail twice
- Sensor Failure
- Backup Sensor is Us

Diagnosability Analysis

compass.py

Trace A
Compliant with diagnosability property.

Name	Step1	Step2	Step3	Step4
alarmF				
alarmS				
filters.filter1.#degrade				
filters.filter1.#die				
filters.filter1.error	OK	OK	OK	OK
filters.filter1.input	1	15	1	15
filters.filter1.mode				
filters.filter1.output	2	30	2	30
filters.filter1._resetEvent				
filters.filter2.#degrade				
filters.filter2.#die				
filters.filter2.error	OK	OK	OK	OK
filters.filter2.input	1	1	1	1
filters.filter2.mode				
filters.filter2.output	2	2	2	2

Trace B
Not compliant with diagnosability property.

Name	Step1	Step2	Step3	Step4
alarmF				
alarmS				
filters.filter1.#degrade				
filters.filter1.#die				
filters.filter1.error	OK	OK	OK	OK
filters.filter1.input	1	2	15	1
filters.filter1.mode				
filters.filter1.output	2	4	30	2
filters.filter1._resetEvent				
filters.filter2.#degrade				
filters.filter2.#die				
filters.filter2.error	OK	OK	OK	OK
filters.filter2.input	1	1	1	1
filters.filter2.mode				
filters.filter2.output	2	2	2	2

Run Check The property **Sensor component fails** was found to be **not diagnosable**.

Industrial Evaluation

Outline

Motivation

COMPASS Project Overview

System Specifications

Error Modelling

Analysis Facilities

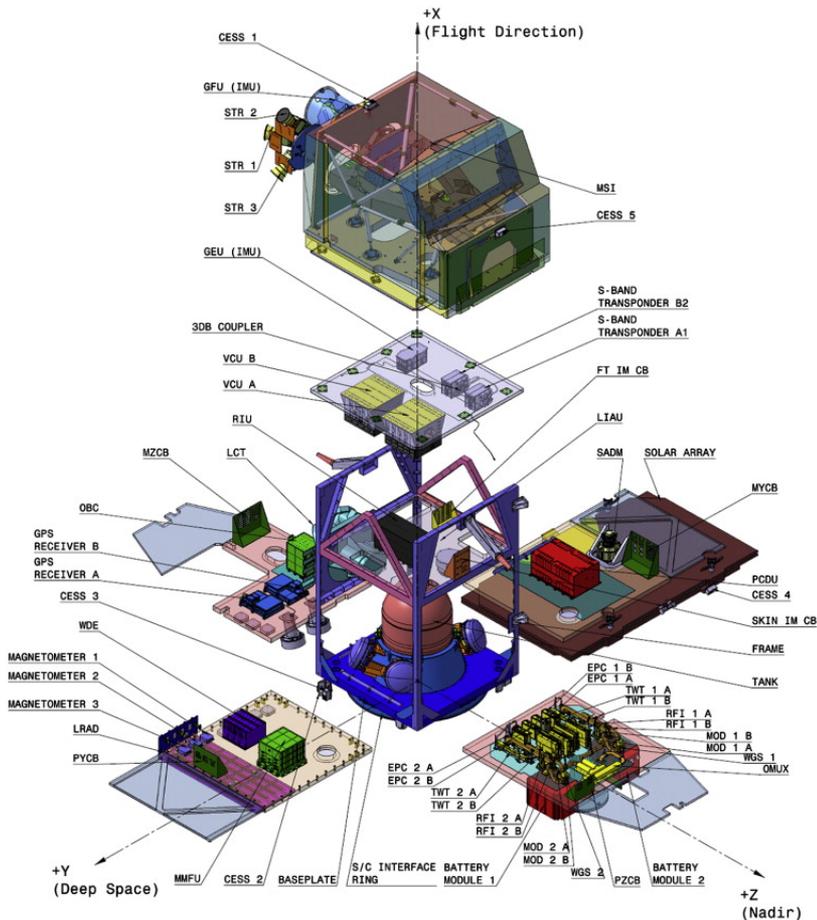
Industrial Evaluation

Conclusion

Case Study: Platform of ESA Satellite

Platform for basic functionality:

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



Note: Shown picture is not from the case study

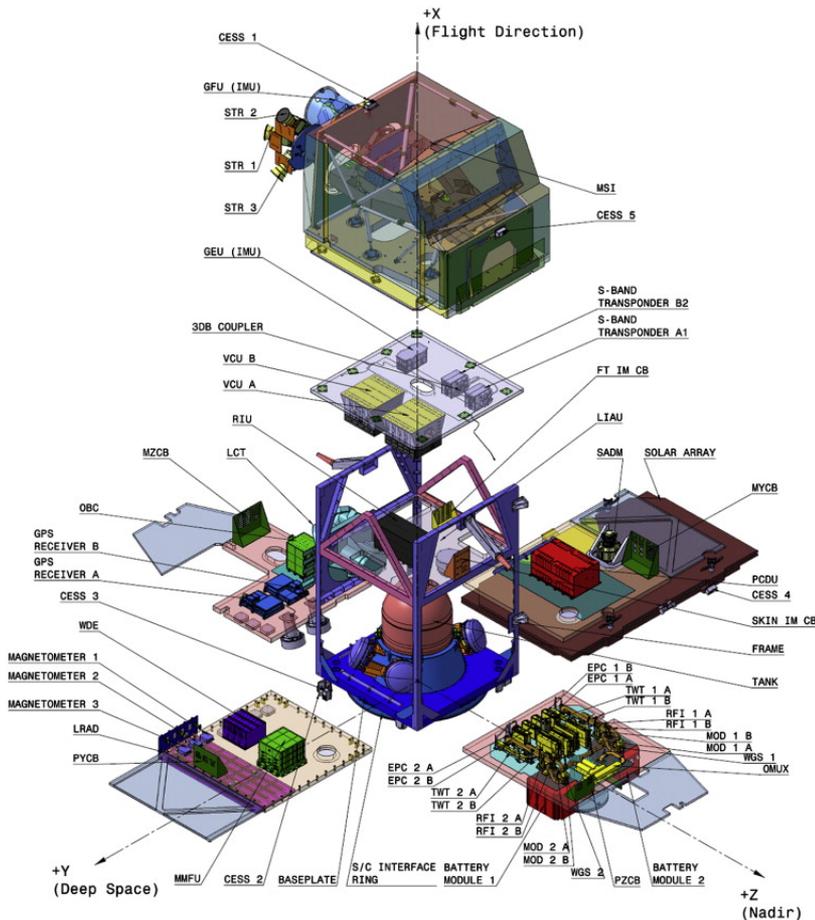
Case Study: Platform of ESA Satellite

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



Note: Shown picture is not from the case study

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

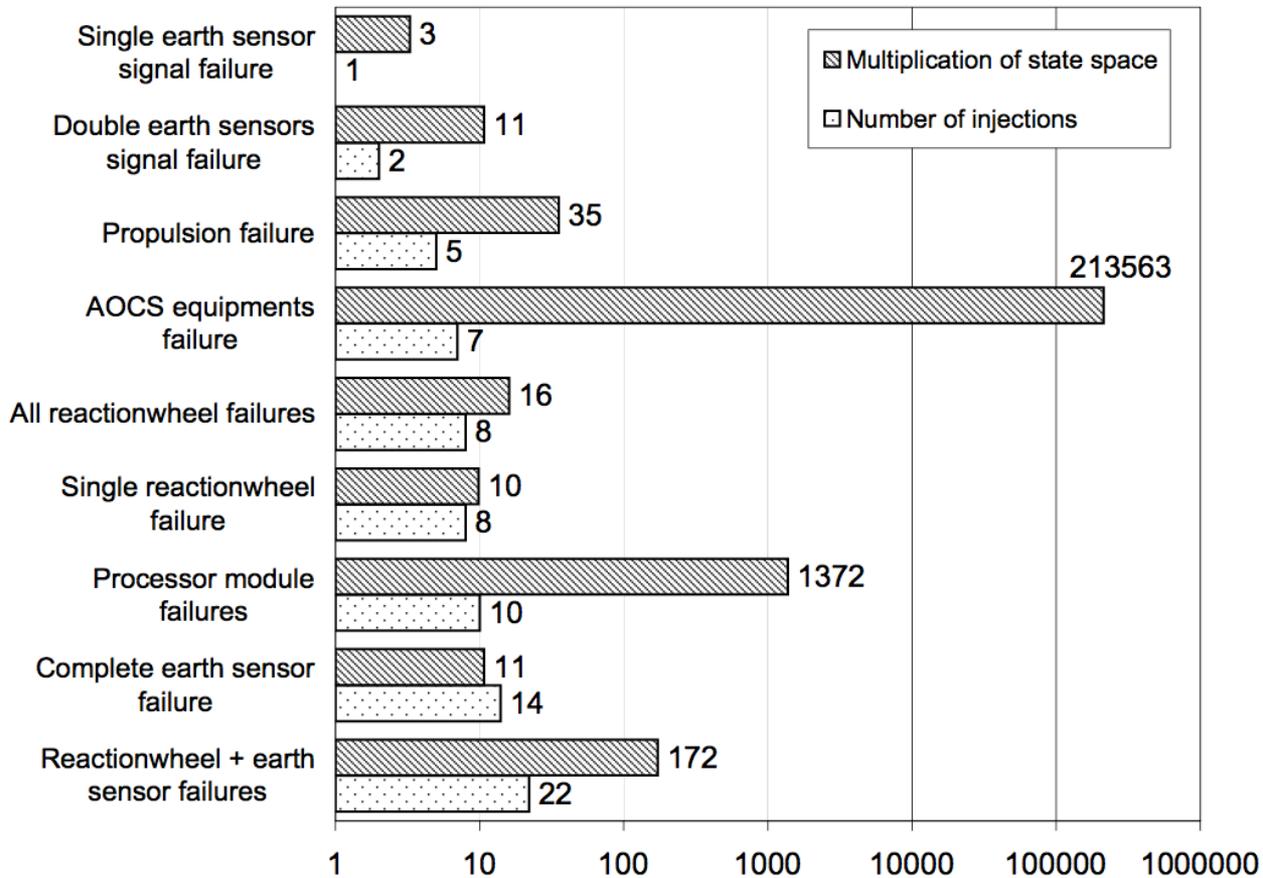
Verification & validation objectives

- 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		

Effect of Fault Injections



Conclusion

Outline

Motivation

COMPASS Project Overview

System Specifications

Error Modelling

Analysis Facilities

Industrial Evaluation

Conclusion

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

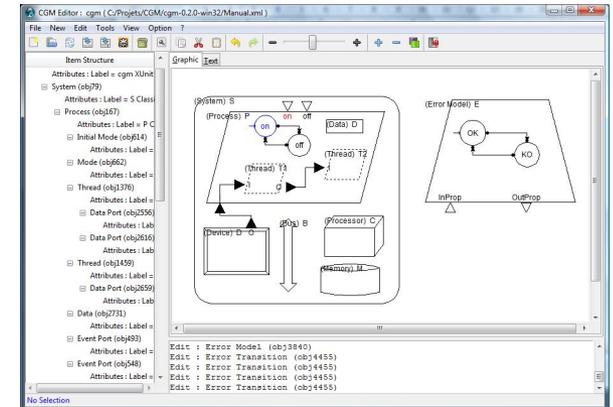
Achievements

- 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)



Conclusion

The COMPASS Web Site

COMPASS
Correctness, Modeling and Performance of Aerospace Systems

[Overview](#) [About](#) [Publications](#) [Tools & Download](#) [FAQ](#) [Workshop](#) [Contact](#)

European Space Agency

RWTH Aachen University

Fondazione Bruno Kessler

Thales Alenia Space

Welcome to COMPASS!

The COMPASS Project is an international research project for developing a theoretical and technological basis for the system-software co-engineering approach focusing on a coherent set of specification and analysis techniques for evaluation of system-level correctness, safety, dependability and performability of on-board computer-based aerospace systems. These techniques shall significantly improve the reliability of modern and future space missions.

The project is running since its kick-off in February 1st, 2008. The consortium consists of the chair of [Software Modeling and Verification](#) at the Institute of Computer Science of the RWTH Aachen University in Germany (project coordinator), research center [Fondazione Bruno Kessler](#) in Italy (research subcontractor), [Thales Alenia Space](#) in France (industrial subcontractor), [European Space Agency](#) (funder). The duration of the project is two years and has a total budget of half million euro.

In the beginning of 2011, an extension of the COMPASS project was commenced for building a graphical drag and drop modeller for the SLIM modelling language. The consortium consisted of RWTH Aachen University, [Ellidiss Technologies](#) and Thales Alenia Space. Total budget is hundred thousand euro and it was completed in December 2011.

For details, please read through the [about](#), [publications](#) and [tools & download](#). Screenshots of COMPASS Toolset 2.0.1 are outlined below.

May, 2013

COMPASS 2.3 and the Graphical Modeller 1.0.9 is now released for everyone in ESA memberstates through the [download section](#).

November, 2012

The COMPASS toolset is a pillar in the technological basis for the recently started [Distributed MILS](#) project funded by the European Union. Aim of D-MILS is to add security aspects to certify compositional and distributed system models.

April, 2012

The SLIM graphical drag-and-drop modeller 1.0.7 developed by Ellidiss Technologies and RWTH Aachen University is available through the download form.

January, 2012

The SLIM graphical drag-and-drop modeller developed by Ellidiss Technologies and RWTH Aachen University has been successfully completed. The manual and user manual are freely available for non-commercial purposes.

Editor: cgm [C:\Project\COM\cgm-0.2.0-win32\Manual.html]

Item Structure

- Label - cgm\Unit
- Attributes: Label - S Class
- Process (obj167)
- Attributes: Label = P C
- Initial Mode (obj124)
- Attributes: Label =
- Mode (obj162)
- Attributes: Label =

Graphic View

(Process) P

(Data) D

(Error Model) E

(Thread) T

(Thread) T

(OK) O

(KO) K

Conclusion

The End

