COMPASS Tutorial at AADL Standards Meeting
Overview of the COMPASS Project

Thomas Noll
Software Modeling and Verification Group

joint work with Marco Bozzano, Alessandro Cimatti, Pierre Dissaux, Joost-Pieter Katoen, Viet Yen Nguyen, Xavier Olive, Marco Roveri, and Yuri Yushtein

October 20, 2011; Toulouse, France
Agenda

1. Overview of the COMPASS Project [Th. Noll]
2. Demo of COMPASS Graphical Modeling Tool [P. Dissaux]
3. Demo of COMPASS Toolset [V.Y. Nguyen]
Contents of COMPASS Overview

1. Introduction and Challenges

2. System Specification
   - Behavioral Modeling
   - Formal Semantics
   - Error Modeling
   - Property Specification

3. Analysis Facilities

4. Industrial Evaluation

5. Conclusions and Outlook
Overview

1 Introduction and Challenges

2 System Specification
   - Behavioral Modeling
   - Formal Semantics
   - Error Modeling
   - Property Specification

3 Analysis Facilities

4 Industrial Evaluation

5 Conclusions and Outlook
Domain: Fault-Tolerant Space System Architectures

ExoMars Rover
- 4 to 21 min. for radio latency to earth
- Martian days autonomous survival
Domain: Fault-Tolerant Space System Architectures

ExoMars Rover
- 4 to 21 min. for radio latency to earth
- Martian days autonomous survival

Autonomous Transfer Vehicle
- human-rated: multiple-failure tolerant design
- autonomous docking with four overrides: HOLD, RETREAT, ESCAPE, ABORT
Extreme Dependability!

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

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

Challenges

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

Limitations

HW verified independently of SW with exaggerated mutual assumptions.
Current Limitations

Limitations

HW verified independently of SW with exaggerated mutual assumptions.

Safety & dependability analyses are isolated from HW/SW models.
Current **Limitations**

**Limitations**

HW verified independently of SW with exaggerated mutual assumptions.

Safety & dependability analyses are isolated from HW/SW models.

Multiple modeling formalisms for different system aspects (e.g. real-time, probabilistic, hybrid).
# Current Limitations

**Limitations**

- HW verified independently of SW with exaggerated mutual assumptions.
- Safety & dependability analyses are isolated from HW/SW models.
- Multiple modeling formalisms for different system aspects (e.g. real-time, probabilistic, hybrid).
- Non-nominal operational modes are overly abstracted to fit various models.
Current Limitations vs. COMPASS Solutions

Solutions

Combine HW, SW and their bindings + ...

error models + ...

real-time, probabilistic and hybrid aspects + ...

non-nominal modes in a single integrated model.
Our Objective

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

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

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

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

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

COMPASS

COrrrectnes, Modeling and Performance of AeroSpace Systems
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

Funding & supervision

- European Space Agency
Approach in a Nutshell

Design a System-Level Integrated Modeling Language (SLIM) based on (core) AADL and its Error Annex [V1].
Approach in a Nutshell

Design a **System-Level Integrated Modeling Language (SLIM)** based on (core) AADL and its Error Annex \[V1\].

Equip this modeling language with a **formal semantics**.
Approach in a Nutshell

Design a System-Level Integrated Modeling Language (SLIM) based on (core) AADL and its Error Annex [V1].

Equip this modeling language with a formal semantics.

Use specification patterns to ease the description of system properties.
Approach in a Nutshell

Design a **System-Level Integrated Modeling Language (SLIM)** based on (core) AADL and its Error Annex [V1].

Equip this modeling language with a **formal semantics**.

Use **specification patterns** to ease the description of system properties.

Support the system-engineering language by powerful **model-checking tools** for correctness, safety, performance and dependability analysis.
Approach in a Nutshell

Design a System-Level Integrated Modeling Language (SLIM) based on (core) AADL and its Error Annex [V1].

Equip this modeling language with a formal semantics.

Use specification patterns to ease the description of system properties.

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

Evaluate their effectiveness by industrial case studies.
COMPASS 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 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: \( \approx 750 \) kEuro; at peak times \( \approx 10 \) programmers involved
Overview

1. Introduction and Challenges

2. System Specification
   - Behavioral Modeling
   - Formal Semantics
   - Error Modeling
   - Property Specification

3. Analysis Facilities

4. Industrial Evaluation

5. Conclusions and Outlook
Methodology
SLIM Example: Redundant Power System

Redundant power system:

- contains two batteries
- power switches from primary to backup mode (and back) when batt1 (batt2) is empty
SLIM Example: Redundant Power System

Redundant power system:

- contains two batteries
- power switches from primary to backup mode (and back) when batt1 (batt2) is empty

We shall show:

- hybrid behavior of the batteries
- composition of the power system
- formalization by automata
- semantics as transition systems
- interweaving of errors
Modeling a Battery in SLIM

Component type and implementation:

device type Battery

end Battery;
device implementation Battery.Imp

end Battery.Imp;
Modeling a Battery in SLIM

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 in SLIM

Adding modes behavior:

```plaintext
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 in SLIM

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;
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 in SLIM

Power system with battery subcomponents:

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

def system implementation Power.Imp
  subcomponents
    batt1: device Battery.Imp
    batt2: device Battery.Imp
end Power.Imp;
```

Thomas Noll
Modeling the Redundant Power System in SLIM

Adding dynamic reconfiguration:

```system
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 in SLIM

Adding **port connections**:

```plaintext
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;
```
SLIM vs. AADL [V1]

Omissions

Many advanced features of AADL such as property associations, component refinement, prototypes, event data ports, in out ports, ...
SLIM vs. AADL [V1]

Omissions

Many 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).
**SLIM vs. AADL [V1]**

### Omissions

Many 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., clocks, mode invariants, trajectory equations
- specification of **observability requirements**
# Event-Data Automata

**Definition (Event-data automaton)**

An event-data automaton (EDA) is a tuple

\[ \mathcal{A} = (M, m_0, X, v_0, \nu, E, \rightarrow) \]

with

- \( M \) finite set of modes
  - \( m_0 \in M \) starting mode
- \( X = IX \uplus OX \uplus LX \) finite set of input/output/local variables
- \( V := \{ v \mid v : X \rightarrow \ldots \} \) valuations
  - \( v_0 \in V \) starting valuation
- \( \nu : M \rightarrow (V \rightarrow \mathbb{B}) \) mode invariants (where \( \nu(m_0, v_0) = \text{true} \))
- \( E = IE \uplus OE \) finite set of input/output events
- \( \rightarrow \subseteq M \times E_\tau \times (V \rightarrow \mathbb{B}) \times (V \rightarrow V) \times M \) (mode) transition relation (where \( E_\tau := E \cup \{ \tau \} \))
Semantics of a SLIM Component

- SLIM modes/invariants/transitions
  \[\leadsto\] EDA modes/invariants/transitions

Example (Battery)

- \( M = \{\text{charged, depleted}\}, \ m_0 = \text{charged} \)
Semantics of a SLIM Component

- SLIM modes/invariants/transitions
  ➞ EDA modes/invariants/transitions
- Incoming/outgoing data ports ➞ input/output variables

Example (Battery)

- \( M = \{\text{charged, depleted}\}, \ m_0 = \text{charged} \)
- \( IX = \emptyset, \ OX = \{\text{voltage}\} \)
Semantics of a SLIM Component

- SLIM modes/invariants/transitions
  $\leadsto$ EDA modes/invariants/transitions
- Incoming/outgoing data ports $\leadsto$ input/output variables
- Data subcomponents $\leadsto$ local variables

Example (Battery)

- $M = \{\text{charged, depleted}\}$, $m_0 = \text{charged}$
- $IX = \emptyset$, $OX = \{\text{voltage}\}$
- $LX = \{\text{energy}\}$
Semantics of a SLIM Component

- SLIM modes/invariants/transitions
  ⟷ EDA modes/invariants/transitions
- Incoming/outgoing data ports ⟷ input/output variables
- Data subcomponents ⟷ local variables
- Incoming/outgoing event ports ⟷ input/output events

Example (Battery)

- \( M = \{ \text{charged}, \text{depleted} \} \), \( m_0 = \text{charged} \)
- \( IX = \emptyset \), \( OX = \{ \text{voltage} \} \)
- \( LX = \{ \text{energy} \} \)
- \( IIE = \emptyset \), \( OE = \{ \text{empty} \} \)
Operational Semantics of EDA

- **States** are pairs: a mode and a variable valuation
- **Transitions**: timed or internal or event-labeled

Example (Battery)

\[
\langle \text{mode} = \text{charged}, \text{energy} = 100.0, \text{voltage} = 6.0 \rangle \downarrow 30.0 \langle \text{mode} = \text{charged}, \text{energy} = 40.0, \text{voltage} = 6.0 \rangle \\
\downarrow \tau \langle \text{voltage} := \cdots \rangle \langle \text{mode} = \text{charged}, \text{energy} = 40.0, \text{voltage} = 4.8 \rangle \\
\downarrow 10.0 \langle \text{mode} = \text{charged}, \text{energy} = 20.0, \text{voltage} = 4.8 \rangle \\
\downarrow \tau \langle \text{voltage} := \cdots \rangle \langle \text{mode} = \text{charged}, \text{energy} = 20.0, \text{voltage} = 4.4 \rangle \\
\downarrow \text{empty} \langle \text{mode} = \text{depleted}, \text{energy} = 20.0, \text{voltage} = 4.4 \rangle \downarrow \cdots
\]
Operational Semantics of EDA

- States are pairs: a mode and a variable valuation
- Transitions: timed or internal or event-labeled

Example (Battery)

\[
\langle \text{mode} = \text{charged}, \text{energy} = 100.0, \text{voltage} = 6.0 \rangle
\]
Operational Semantics of EDA

- States are pairs: a mode and a variable valuation
- Transitions: timed or internal or event-labeled

Example (Battery)

\[
\langle \text{mode} = \text{charged}, \text{energy} = 100.0, \text{voltage} = 6.0 \rangle \\
\downarrow 30.0 \\
\langle \text{mode} = \text{charged}, \text{energy} = 40.0, \text{voltage} = 6.0 \rangle
\]
Operational Semantics of EDA

- States are pairs: a mode and a variable valuation
- Transitions: timed or internal or event-labeled

Example (Battery)

\[
\langle \text{mode} = \text{charged}, \text{energy} = 100.0, \text{voltage} = 6.0 \rangle \\
\downarrow 30.0 \\
\langle \text{mode} = \text{charged}, \text{energy} = 40.0, \text{voltage} = 6.0 \rangle \\
\downarrow \tau \langle \text{voltage}:=\ldots \rangle \\
\langle \text{mode} = \text{charged}, \text{energy} = 40.0, \text{voltage} = 4.8 \rangle
\]
Operational Semantics of EDA

- States are pairs: a mode and a variable valuation
- Transitions: timed or internal or event-labeled

Example (Battery)

\[
\begin{align*}
\langle \text{mode} = \text{charged, energy} = 100.0, \text{voltage} = 6.0 \rangle & \quad \downarrow 30.0 \\
\langle \text{mode} = \text{charged, energy} = 40.0, \text{voltage} = 6.0 \rangle & \quad \downarrow \tau \langle \text{voltage:=...} \rangle \\
\langle \text{mode} = \text{charged, energy} = 40.0, \text{voltage} = 4.8 \rangle & \quad \downarrow 10.0 \\
\langle \text{mode} = \text{charged, energy} = 20.0, \text{voltage} = 4.8 \rangle &
\end{align*}
\]
Operational Semantics of EDA

- States are pairs: a mode and a variable valuation
- Transitions: timed or \texttt{internal} or event-labeled

Example (Battery)

\[
\langle \text{mode} = \text{charged}, \text{energy} = 100.0, \text{voltage} = 6.0 \rangle \\
\downarrow 30.0 \\
\langle \text{mode} = \text{charged}, \text{energy} = 40.0, \text{voltage} = 6.0 \rangle \\
\downarrow \tau \langle \text{voltage}:=\ldots \rangle \\
\langle \text{mode} = \text{charged}, \text{energy} = 40.0, \text{voltage} = 4.8 \rangle \\
\downarrow 10.0 \\
\langle \text{mode} = \text{charged}, \text{energy} = 20.0, \text{voltage} = 4.8 \rangle \\
\downarrow \tau \langle \text{voltage}:=\ldots \rangle \\
\langle \text{mode} = \text{charged}, \text{energy} = 20.0, \text{voltage} = 4.4 \rangle
\]
## Operational Semantics of EDA

- States are pairs: a mode and a variable valuation
- Transitions: timed or internal or event-labeled

### Example (Battery)

\[
\begin{align*}
\langle \text{mode} = \text{charged}, \text{energy} = 100.0, \text{voltage} = 6.0 \rangle \\
\downarrow 30.0
\end{align*}
\]

\[
\begin{align*}
\langle \text{mode} = \text{charged}, \text{energy} = 40.0, \text{voltage} = 6.0 \rangle \\
\downarrow \tau\langle \text{voltage}:=\ldots \rangle
\end{align*}
\]

\[
\begin{align*}
\langle \text{mode} = \text{charged}, \text{energy} = 40.0, \text{voltage} = 4.8 \rangle \\
\downarrow 10.0
\end{align*}
\]

\[
\begin{align*}
\langle \text{mode} = \text{charged}, \text{energy} = 20.0, \text{voltage} = 4.8 \rangle \\
\downarrow \tau\langle \text{voltage}:=\ldots \rangle
\end{align*}
\]

\[
\begin{align*}
\langle \text{mode} = \text{charged}, \text{energy} = 20.0, \text{voltage} = 4.4 \rangle \\
\downarrow \text{empty}
\end{align*}
\]

\[
\begin{align*}
\langle \text{mode} = \text{depleted}, \text{energy} = 20.0, \text{voltage} = 4.4 \rangle
\end{align*}
\]
**Operational Semantics of EDA**

- States are pairs: a mode and a variable valuation
- Transitions: timed or internal or event-labeled

**Example (Battery)**

<table>
<thead>
<tr>
<th>State</th>
<th>Mode</th>
<th>Energy</th>
<th>Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>charged</td>
<td>100.0</td>
<td>6.0</td>
</tr>
<tr>
<td>2</td>
<td>charged</td>
<td>40.0</td>
<td>6.0</td>
</tr>
<tr>
<td>3</td>
<td>charged</td>
<td>40.0</td>
<td>4.8</td>
</tr>
<tr>
<td>4</td>
<td>charged</td>
<td>20.0</td>
<td>4.8</td>
</tr>
<tr>
<td>5</td>
<td>charged</td>
<td>20.0</td>
<td>4.4</td>
</tr>
<tr>
<td>6</td>
<td>depleted</td>
<td>20.0</td>
<td>4.4</td>
</tr>
</tbody>
</table>

- Transition 1: \( \downarrow 30.0 \)
- Transition 2: \( \downarrow \tau \langle \text{voltage:=...} \rangle \)
- Transition 3: \( \downarrow 10.0 \)
- Transition 4: \( \downarrow \tau \langle \text{voltage:=...} \rangle \)
- Transition 5: \( \downarrow \text{empty} \)
- Transition 6: \( \downarrow \ldots \)
Networks of Event-Data Automata

Dynamic reconfiguration

⇒ component activity and port connections mode dependent

Definition (Networks of Event-Data Automata)

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

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

with \( n \geq 1 \), \([n] := \{1, \ldots, n\}\), and

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

- SLIM subcomponent declarations $\rightsquigarrow$ activation mapping:
  - root component always active
  - $c$ active and in mode $m$, subcomponent $c'$ of $c$ activated in $m$ $\implies c'$ active

Example (Power System)

For Power/Battery1/Battery2 ($m_1, m_2 \in \{\text{charged, depleted}\}$):

- $\alpha(\text{primary}, m_1, m_2) = \{1, 2\}$
- $\alpha(\text{backup}, m_1, m_2) = \{1, 3\}$
Semantics of an Entire SLIM Model

- SLIM event/data connections $\leadsto EC/DC$ mappings:
  follow all end-to-end chains of port connections

Example (Power System)

For Power/Battery1/Battery2 ($m_1, m_2 \in \{\text{charged, depleted}\}$):

- $EC(\text{primary}, m_1, m_2) = \{(2.\text{empty}, 1.\text{batt1.empty})\}$
- $EC(\text{backup}, m_1, m_2) = \{(3.\text{empty}, 1.\text{batt2.empty})\}$
- $DC(\text{primary}, m_1, m_2) = \{(2.\text{voltage}, 1.\text{voltage})\}$
- $DC(\text{backup}, m_1, m_2) = \{(3.\text{voltage}, 1.\text{voltage})\}$
Operational Semantics of Networks of EDAs

- **States**: \((M_1 \times V_1) \times \ldots \times (M_n \times V_n)\)
- **Transitions** determined by active EDAs:
  1. Perform local transitions:
     - timed local transition in all EDAs or
     - internal transition in EDA or
     - multi-way event communication from EDA to \(\geq 1\) connected EDAs
  2. Initialize (re-)activated subcomponents
  3. Establish consistency w.r.t. DC (copy source \(\rightarrow\) target data port)
Operational Semantics of Networks of EDAs

- States: \((M_1 \times V_1) \times \ldots \times (M_n \times V_n)\)
- Transitions determined by active EDAs:
  1. Perform local transitions:
     - timed local transition in all EDAs or
     - internal transition in EDA or
     - multi-way event communication from EDA to \(\geq 1\) connected EDAs
  2. Initialize (re-)activated subcomponents
  3. Establish consistency w.r.t. \(DC\) (copy source \(\rightarrow\) target data port)

Example (Power system)

\[
\langle m={\underline{primary}}, v=6.0 \rangle \| \langle m={\underline{charged}}, e=100.0, v=6.0 \rangle \| \langle m={\underline{charged}}, e=100.0, v=6.0 \rangle
\]
Operational Semantics of Networks of EDAs

- States := \((M_1 \times V_1) \times \ldots \times (M_n \times V_n)\)

- Transitions determined by active EDAs:
  1. Perform local transitions:
     - timed local transition in all EDAs or
     - internal transition in EDA or
     - multi-way event communication from EDA to \(\geq 1\) connected EDAs
  2. Initialize (re-)activated subcomponents
  3. Establish consistency w.r.t. \(DC\) (copy source \(\rightarrow\) target data port)

Example (Power system)

\[
\begin{align*}
\langle m = \underline{primary}, v = 6.0 \rangle &| \langle m = \underline{charged}, e = 100.0, v = 6.0 \rangle | \langle m = \underline{charged}, e = 100.0, v = 6.0 \rangle \\
&\downarrow 40.0 \\
\langle m = primary, v = 6.0 \rangle &| \langle m = \underline{charged}, e = 20.0, v = 6.0 \rangle | \langle m = charged, e = 100.0, v = 6.0 \rangle
\end{align*}
\]
Operational Semantics of Networks of EDAs

- States := \( (M_1 \times V_1) \times \ldots \times (M_n \times V_n) \)

- Transitions determined by active EDAs:
  1. Perform local transitions:
     - timed local transition in all EDAs or
     - internal transition in EDA or
     - multi-way event communication from EDA to \( \geq 1 \) connected EDAs
  2. Initialize (re-)activated subcomponents
  3. Establish consistency w.r.t. \( DC \) (copy source \( \rightarrow \) target data port)

Example (Power system)

\[
\langle m = \underline{primary}, v = 6.0 \rangle \mid \langle m = \underline{charged}, e = 100.0, v = 6.0 \rangle \mid \langle m = \underline{charged}, e = 100.0, v = 6.0 \rangle
\]
\( \Downarrow 40.0 \)

\[
\langle m = \underline{primary}, v = 6.0 \rangle \mid \langle m = \underline{charged}, e = 20.0, v = 6.0 \rangle \mid \langle m = \underline{charged}, e = 100.0, v = 6.0 \rangle
\]
\( \Downarrow \tau\langle \text{voltage} := \ldots \rangle \)

\[
\langle m = \underline{primary}, v = 4.4 \rangle \mid \langle m = \underline{charged}, e = 20.0, v = 4.4 \rangle \mid \langle m = \underline{charged}, e = 100.0, v = 6.0 \rangle
\]
Operational Semantics of Networks of EDAs

- States := \((M_1 \times V_1) \times \ldots \times (M_n \times V_n)\)
- Transitions determined by active EDAs:
  1. Perform local transitions:
     - timed local transition in all EDAs or
     - internal transition in EDA or
     - multi-way event communication from EDA to \(\geq 1\) connected EDAs
  2. Initialize (re-)activated subcomponents
  3. Establish consistency w.r.t. \(DC\) (copy source \(\rightarrow\) target data port)

Example (Power system)

\[
\begin{align*}
\langle m_{\text{primary}}, v = 6.0 \rangle & \quad \langle m_{\text{charged}}, e = 100.0, v = 6.0 \rangle \quad \langle m_{\text{charged}}, e = 100.0, v = 6.0 \rangle \\
\downarrow 40.0 \\
\langle m_{\text{primary}}, v = 6.0 \rangle & \quad \langle m_{\text{charged}}, e = 20.0, v = 6.0 \rangle \quad \langle m_{\text{charged}}, e = 100.0, v = 6.0 \rangle \\
\downarrow \tau \langle \text{voltage:=...} \rangle \\
\langle m_{\text{primary}}, v = 4.4 \rangle & \quad \langle m_{\text{charged}}, e = 20.0, v = 4.4 \rangle \quad \langle m_{\text{charged}}, e = 100.0, v = 6.0 \rangle \\
\downarrow \tau \langle \text{empty} \rangle \\
\langle m_{\text{backup}}, v = 6.0 \rangle & \quad \langle m_{\text{depleted}}, e = 20.0, v = 4.4 \rangle \quad \langle m_{\text{charged}}, e = 100.0, v = 6.0 \rangle
\end{align*}
\]
Operational Semantics of Networks of EDAs

- States := \((M_1 \times V_1) \times \ldots \times (M_n \times V_n)\)
- Transitions determined by active EDAs:
  1. Perform local transitions:
     - timed local transition in all EDAs or
     - internal transition in EDA or
     - multi-way event communication from EDA to \(\geq 1\) connected EDAs
  2. Initialize (re-)activated subcomponents
  3. Establish consistency w.r.t. \(DC\) (copy source \(\rightarrow\) target data port)

Example (Power system)

\[
\begin{align*}
\langle m = \text{primary}, v = 6.0 \rangle &\quad | \quad \langle m = \text{charged}, e = 100.0, v = 6.0 \rangle \quad | \quad \langle m = \text{charged}, e = 100.0, v = 6.0 \rangle \\
\downarrow 40.0 \\
\langle m = \text{primary}, v = 6.0 \rangle &\quad | \quad \langle m = \text{charged}, e = 20.0, v = 6.0 \rangle \quad | \quad \langle m = \text{charged}, e = 100.0, v = 6.0 \rangle \\
\downarrow \tau \langle \text{voltage} := \ldots \rangle \\
\langle m = \text{primary}, v = 4.4 \rangle &\quad | \quad \langle m = \text{charged}, e = 20.0, v = 4.4 \rangle \quad | \quad \langle m = \text{charged}, e = 100.0, v = 6.0 \rangle \\
\downarrow \tau \langle \text{empty} \rangle \\
\langle m = \text{backup}, v = 6.0 \rangle &\quad | \quad \langle m = \text{depleted}, e = 20.0, v = 4.4 \rangle \quad | \quad \langle m = \text{charged}, e = 100.0, v = 6.0 \rangle \\
\downarrow 40.0 \\
\langle m = \text{backup}, v = 6.0 \rangle &\quad | \quad \langle m = \text{depleted}, e = 20.0, v = 4.4 \rangle \quad | \quad \langle m = \text{charged}, e = 20.0, v = 6.0 \rangle
\end{align*}
\]
Operational Semantics of Networks of EDAs

- States := $(M_1 \times V_1) \times \ldots \times (M_n \times V_n)$
- Transitions determined by active EDAs:
  1. Perform local transitions:
     - timed local transition in all EDAs or
     - internal transition in EDA or
     - multi-way event communication from EDA to $\geq 1$ connected EDAs
  2. Initialize (re-)activated subcomponents
  3. Establish consistency w.r.t. $DC$ (copy source $\rightarrow$ target data port)

Example (Power system)

<table>
<thead>
<tr>
<th>States</th>
<th>Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle m_{primary}, v=6.0 \rangle$</td>
<td>$\downarrow 40.0$</td>
</tr>
<tr>
<td>$\langle m_{primary}, v=6.0 \rangle$</td>
<td>$\langle m_{charged}, e=20.0, v=6.0 \rangle$</td>
</tr>
<tr>
<td>$\langle m_{primary}, v=4.4 \rangle$</td>
<td>$\langle m_{charged}, e=20.0, v=4.4 \rangle$</td>
</tr>
<tr>
<td>$\langle m_{backup}, v=6.0 \rangle$</td>
<td>$\langle m_{depleted}, e=20.0, v=4.4 \rangle$</td>
</tr>
<tr>
<td>$\langle m_{backup}, v=6.0 \rangle$</td>
<td>$\langle m_{depleted}, e=20.0, v=4.4 \rangle$</td>
</tr>
</tbody>
</table>

Example (Power system)

$\langle m_{primary}, v=6.0 \rangle$ $\downarrow 40.0$
$\langle m_{primary}, v=6.0 \rangle$ $\downarrow \tau\{voltage:=\ldots\}$
$\langle m_{primary}, v=4.4 \rangle$ $\downarrow \tau\{empty\}$
$\langle m_{backup}, v=6.0 \rangle$ $\downarrow 40.0$
$\langle m_{backup}, v=6.0 \rangle$ $\downarrow \ldots$
Integrating Erroneous and Nominal Behavior

Nominal Model = SLIM components

Error Models

Fault Injections

Automatic Model Extension

Extended Model = nominal + error effects + degraded 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**

```plaintext
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 subcomponent and \(a\) the fault effect.
Error Modeling

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

test events
end BatteryFailure.Imp;

Fault injection example
In error state dead, voltage := 0
Model Extension

Nominal model + error model + fault injections = extended model

- Modes are **pairs** of nominal modes and error model states
  - starting mode := (original starting mode, starting error state)
- Event ports += error propagations
- Event port connections += propagation port connections
- Transition relation := all possible **interleavings and interactions** between nominal and error model, taking failure effects into account
- Other elements (e.g., mode invariants) are unaffected
Model Extension

Nominal model $+$ error model $+$ fault injections $=$ \textit{extended model}

- Modes are \textit{pairs} of nominal modes and error model states
  - starting mode $:=$ (original starting mode, starting error state)
- Event ports $+:=$ error propagations
- Event port connections $+:=$ propagation port connections
- Transition relation $:=$ all possible \textit{interleavings and interactions} between nominal and error model, taking failure effects into account
- Other elements (e.g., mode invariants) are unaffected

\textbf{Probabilistic error transitions}

As an error model has probabilistic transitions, our semantical model has to be equipped with such transitions.

This yields \textit{interactive Markov chains} $:=$ LTS $+$ Markov chains.
Battery Component

Nominal specification:

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 ...;
    depleted: mode while ...;
  transitions
    charged -[then voltage:=...]-> charged;
    charged -[empty when energy<=20.0]-> depleted;
    depleted -[then voltage:=...]-> depleted;

end Battery.Imp;
Battery Component After Model Extension

Product construction for modes:

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#ok: activation mode while ...;
    depleted#ok, charged#dead, depleted#dead: mode while ...;
  transitions
    charged -[then voltage:=...]-> charged;
    charged -[empty when energy<=20.0]-> depleted;
    depleted -[then voltage:=...]-> depleted;
  end Battery.Imp;
Battery Component After Model Extension

Integrate nominal transitions:

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#ok: activation mode while ...;
    depleted#ok, charged#dead, depleted#dead: mode while ...;
  transitions
    charged#ok -[then voltage:=...]-> charged#ok;
    charged#ok -[empty when energy<=20.0]-> depleted#ok;
    depleted#ok -[then voltage:=...]-> depleted#ok;

end Battery.Imp;
Battery Component After Model Extension

Fault injection:

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#ok: activation mode while ...;
  depleted#ok, charged#dead, depleted#dead: mode while ...;
transitions
  charged#ok -[then voltage:=...]-> charged#ok;
  charged#ok -[empty when energy<=20.0]-> depleted#ok;
  depleted#ok -[then voltage:=...]-> depleted#ok;
  charged#ok -[prob 0.001 then voltage:=0]-> charged#dead;
  depleted#ok -[prob 0.001 then voltage:=0]-> depleted#dead;
end Battery.Imp;
Battery Component After Model Extension

Nominal transitions with fault effects:

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

device implementation Battery.Imp
  subcomponents
    energy: data continuous default 100.0;
  modes
    charged#ok: activation mode while ...;
    depleted#ok, charged#dead, depleted#dead: mode while ...;
  transitions
    charged#ok -[then voltage:=...]-> charged#ok;
    charged#ok -[empty when energy<=20.0]-> depleted#ok;
    depleted#ok -[then voltage:=...]-> depleted#ok;
    charged#ok -[prob 0.001 then voltage:=0]-> charged#dead;
    depleted#ok -[prob 0.001 then voltage:=0]-> depleted#dead;
    charged#dead -[then voltage:=0]-> charged#dead;
    charged#dead -[empty when energy<=20.0]-> depleted#dead;
    depleted#dead -[then voltage:=0]-> depleted#dead;

end Battery.Imp;
Battery Component After Model Extension

Add error propagations:

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

device implementation Battery.Imp
  subcomponents
    energy: data continuous default 100.0;
  modes
    charged#ok: activation mode while ...;
    depleted#ok, charged#dead, depleted#dead: mode while ...;
  transitions
    charged#ok -[then voltage:=...]-> charged#ok;
    charged#ok -[empty when energy<=20.0]-> depleted#ok;
    depleted#ok -[then voltage:=...]-> charged#ok;
    charged#ok -[prob 0.001 then voltage:=0]-> charged#dead;
    depleted#ok -[prob 0.001 then voltage:=0]-> depleted#dead;
    charged#dead -[then voltage:=0]-> charged#dead;
    charged#dead -[empty when energy<=20.0]-> depleted#dead;
    depleted#dead -[then voltage:=0]-> depleted#dead;
    depleted#dead -[batteryDied]-> depleted#dead;
  end Battery.Imp;
The Complete Power System Model

[Device] Battery.imp: batt1

Nominal
- charged
  - energy' = -0.02
  - energy >= 15
  - voltage := f(energy)
- empty
  - energy < 20
- depleted
  - energy' = -0.03
  - voltage := f(energy)

Error
- ok

Data
- energy
- init 100

[Device] Battery.imp: batt2

Nominal
- charged
  - energy' = -0.02
  - energy >= 15
  - voltage := f(energy)
- empty
  - energy < 20
- depleted
  - energy' = -0.03
  - voltage := f(energy)

Error
- ok

Data
- energy
- init 100
Specifying Observability

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

Example:

```plaintext
system PowerSystem
  features
    voltage: out data port real;
    alarm: out data port bool default 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;
```
**Property Specification: Patterns, not Formulas!**

**Patterns**

- The system shall have a behavior where \( x \leq \text{voltage} \leq y \) 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 continuously within time bound \([0,10]\).
Property Specification: Patterns, not Formulas!

Patterns

- The system shall have a behavior where $x \leq \text{voltage} \leq y$ 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 continuously within time bound $[0,10]$.

(by automatic transformation)

Logic

- $\Box(x \leq \text{voltage} \leq y)$  
  (Linear Temporal Logic)
- $\mathcal{P}^{>0.98}[\Box[0,10](\text{voltage} \geq 80)]$  
  (Continuous Stochastic Logic)
**Property Specification: Patterns, not Formulas!**

### Logic

- □(x ≤ voltage ≤ y)  
  (Linear Temporal Logic)
- P > 0.98[□[0,10](voltage ≥ 80)]  
  (Continuous Stochastic Logic)

### Implemented pattern systems

<table>
<thead>
<tr>
<th>Formalism</th>
<th>Intended use</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTL, LTL</td>
<td>functional properties</td>
<td>[Dwyer et al., 1999]</td>
</tr>
<tr>
<td>MTL, TCTL</td>
<td>real-time properties</td>
<td>[Konrad &amp; Cheng, 2005]</td>
</tr>
<tr>
<td>PCTL, CSL</td>
<td>probabilistic properties</td>
<td>[Grunske, 2008]</td>
</tr>
</tbody>
</table>
Overview

1. Introduction and Challenges

2. System Specification
   - Behavioral Modeling
   - Formal Semantics
   - Error Modeling
   - Property Specification

3. Analysis Facilities

4. Industrial Evaluation

5. Conclusions and Outlook
Types of Analyses

1. Validation
   ▶ check logical consistency of logical specification

2. Model checking
   ▶ property patterns, BMC, BDD-based MC, SMT for hybrid

3. Safety and dependability
   ▶ FMEA (impact analysis of fault modes), dynamic FTA

4. Diagnosability
   ▶ FDIR

5. Performance evaluation
   ▶ using probabilistic model checking
   ▶ effective model reduction techniques
## Toolset Components

### NuSMV
- Symbolic LTL and CTL model checker
- BDD- and SAT-based model checking
- Counterexample generation

### RAT
- Requirements analyzer
- Checks logical consistency

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

### SigRef
- (MT)BDD bisimulation minimization
- Models: Markov chains
Tool Architecture

AADL Model
   Compiler
      SMV
         NuSMV

Fault Injections
   Requirement Patterns
      LTL/CTL
      CSL

Validation
   - Consistency
   - Simulation

Correctness
   - Verification (discrete/hybrid)
   - Simulation

Diagnosability
   - FDIR
   - Observability

Safety
   - Dynamic Fault Trees
   - FMEA tables

Performability
   - Probabilistic DFT
   - Performability measures
Model Checking View

![Model Checking View](image)

- **Properties**
  - **Name**: observe output
  - **Always output is**

- **Model Checking**
  - **Model Simulation**

- **Model Checker Options**:
  - Use BDD (CTL and LTL)
  - Use SAT (LTL only)
  - SAT Bound: 10

- **The property is false**
  - The LTL property: $\text{G} \neg \text{output}$ has been found **false**. A counter-example is shown below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Step1</th>
<th>Step2</th>
<th>Step3</th>
<th>Step4</th>
<th>Step5</th>
<th>Step6</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>init</td>
<td>gone_rnd2</td>
<td>gone_rnd12</td>
<td>gone_bit2</td>
<td>gone_bit12</td>
<td>go</td>
</tr>
<tr>
<td>run</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>rnd1.output</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>rnd2.output</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Simulator View

You selected $\%$ as constraints for the simulation.
Simulation current length is $\%$.

Run Model Simulation  Model extended by Fault Injections

Simulation Options:
Length: 10

Simulation
A simulation exists and it is shown by the following trace

<table>
<thead>
<tr>
<th>Name</th>
<th>Step1</th>
<th>Step2</th>
<th>Step3</th>
<th>Step4</th>
<th>Step5</th>
<th>Step6</th>
<th>Step7</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensors.switch</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>filters.switch</td>
<td>Primary</td>
<td>Primary</td>
<td>Primary</td>
<td>Primary</td>
<td>Backup</td>
<td>Backup</td>
<td>Backup</td>
</tr>
<tr>
<td>sensors.mode</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>sensors.sensor1.output</td>
<td>OK</td>
<td>Glitched</td>
<td>OK</td>
<td>Dead</td>
<td>Dead</td>
<td>Dead</td>
<td>Dead</td>
</tr>
<tr>
<td>sensors.sensor1.error</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>Glitched</td>
<td>OK</td>
</tr>
<tr>
<td>sensors.sensor2.output</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>sensors.sensor2.error</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>filters.mode</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>filters.filter1.output</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>filters.filter1.error</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>filters.filter2.output</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>filters.filter2.error</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>value.mode</td>
<td>multInitialMode</td>
<td>multInitialMode</td>
<td>multInitialMode</td>
<td>multInitialMode</td>
<td>multInitialMode</td>
<td>multInitialMode</td>
<td>multInitialMode</td>
</tr>
</tbody>
</table>
Performance View

Please select one property on the left to check performability.

The probability is: 65.32801 %

Graph showing the probability over time.
Overview

1. Introduction and Challenges

2. System Specification
   - Behavioral Modeling
   - Formal Semantics
   - Error Modeling
   - Property Specification

3. Analysis Facilities

4. Industrial Evaluation

5. Conclusions and Outlook
1st Case Study: Satellite Thermal Regulation

Challenges:

- Hardware (sensors, heaters) and software (control) co-engineering
- Hybrid behavior (temperatures)
- Dynamic reconfiguration (redundancy)
- State-space explosion
2nd Case Study: Satellite FDIR System

Goal
Assess effectiveness of FDIR measures

Model components:

- satellite mode management during transfer-to-orbit phase
- AOCS (Attitude and Orbit Control System) mode management
- abstraction of AOCS equipment (sensors, gyroscope, ...)
- FDIR action sequence
# 2nd Case Study: Satellite FDIR System

## Goal

Assess effectiveness of FDIR measures

## Model components:

- satellite mode management during transfer-to-orbit phase
- AOCS (Attitude and Orbit Control System) mode management
- abstraction of AOCS equipment (sensors, gyroscope, ...)
- FDIR action sequence

## Analysis problems:

- identification of failures leading to a given FDIR level
- identification of failures entailing a system reconfiguration
- impact of reconfiguration on satellite and AOCS mode
Scalability

Verification Time

Seconds

Degree of Redundancy

NuSMV
SigRef
MRMC

0
7500
15000
22500
30000

2 3 4 5 6 7 8 9 10

Thomas Noll
3rd Case Study: Platform of Satellite

Launches between 2012-2020

**Payload** is mission-specific equipment, e.g.:
- telecom transponders,
- navigation signals,
- earth observation telemetry (weather, radiation, salinity).

**Platform** keeps the satellite orbiting in space, consists of:
- attitude & orbital control
- power distribution
- data handling
- communications
- thermal regulation
SLIM Model of Satellite Platform

Verification & validation objectives

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

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

**Model characteristics**
- Functional
- Probabilistic
- Real-time
- Hybrid
**SLIM Model of Satellite Platform**

**Verification & validation objectives**

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

**Model characteristics**

- Functional
- Probabilistic
- Real-time
- Hybrid

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Components</th>
<th>Modes</th>
<th>Faults</th>
<th>Recoveries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional</td>
<td>Components</td>
<td>Modes</td>
<td>Faults</td>
<td>Recoveries</td>
</tr>
<tr>
<td>Probabilistic</td>
<td>Components</td>
<td>Modes</td>
<td>Faults</td>
<td>Recoveries</td>
</tr>
<tr>
<td>Real-time</td>
<td>Components</td>
<td>Modes</td>
<td>Faults</td>
<td>Recoveries</td>
</tr>
<tr>
<td>Hybrid</td>
<td>Components</td>
<td>Modes</td>
<td>Faults</td>
<td>Recoveries</td>
</tr>
</tbody>
</table>

- Components: 99
- Modes: 217
- Faults: 21
- Recoveries: 9

State space of nominal behavior: 48,421,100 states
SLIM Model of Satellite Platform

Verification & validation objectives

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

Model characteristics

- ✓ Functional
- ✓ Probabilistic
- ✓ Real-time
- ✓ Hybrid

Components: 99
Modes: 217
Faults: 21
Recoveries: 9

State space of nominal behavior: 48,421,100 states
SLIM Model of Satellite Platform

Verification & validation objectives

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

Model characteristics

- Functional
- Probabilistic
- Real-time
- Hybrid

Components: 99
Modes: 217
Faults: 21
Recoveries: 9

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

Requirement metrics

- Functional properties: 32
- Probabilistic properties: 2
## Analysis Results

Setup: Intel Xeon 2.33 GHz machine with 16 GB RAM.
Fault injections: earth sensor failure.

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Time (min:sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deadlock checking</td>
<td>3:41</td>
</tr>
<tr>
<td>Model checking “recovers from sensor failure”</td>
<td>10:34</td>
</tr>
<tr>
<td>Fault tree analysis “sensor reading incorrect”</td>
<td>6:35</td>
</tr>
<tr>
<td>Fault tree evaluation “sensor reading incorrect”</td>
<td>0:02</td>
</tr>
<tr>
<td>Fault tolerance evaluation</td>
<td>0:06</td>
</tr>
<tr>
<td>Dynamic fault tree analysis “sensor reading incorrect”</td>
<td>10:51</td>
</tr>
<tr>
<td>Dynamic fault tree evaluation “sensor reading incorrect”</td>
<td>0:03</td>
</tr>
<tr>
<td>Fault detection “sensor failed”</td>
<td>22:47</td>
</tr>
<tr>
<td>Fault isolation “sensor failed”</td>
<td>4:59</td>
</tr>
<tr>
<td>Fault recovery “sensor failed”</td>
<td>10:24</td>
</tr>
<tr>
<td>FMEA “sensor failed”</td>
<td>18:10</td>
</tr>
<tr>
<td>Performability</td>
<td>&gt; 552:00(^1)</td>
</tr>
<tr>
<td>Diagnosability</td>
<td>&gt; 3823:00(^2)</td>
</tr>
</tbody>
</table>

\(^1\) ran out of memory
\(^2\) aborted after 63 hours
Experiences so far

+ Abstraction level of models is appropriate
  ▶ mode transition systems, *not* source code
Experiences so far

+ Abstraction level of models is appropriate
  ▶ mode transition systems, *not* source code

+ Support of incremental approach to system design
  ▶ start with abstract functional representation
  ▶ refinement without breaking structure of model
  ▶ separation of component interface and implementation

− Missing (automatic) link between SLIM and engineering models (UML, Simulink)
  ▶ integration into engineering tool suite

− Tool set performance on timed and hybrid systems needs further study
Experiences so far

- **Abstraction level** of models is appropriate
  - mode transition systems, *not* source code
- **Support of incremental approach** to system design
  - start with abstract functional representation
  - refinement without breaking structure of model
  - separation of component interface and implementation
- **Valuable feedback** from analysis to system designer
  - found several design inconsistencies in satellite case study
  - better understanding of system behavior under (multiple) failures
  - improved safety analysis due to automatic FMEA/FT generation

- Missing (automatic) link between SLIM and engineering models
  - integration into engineering tool suite
- Tool set performance on timed and hybrid systems needs further study
Experiences so far

+ Abstraction level of models is appropriate
  - mode transition systems, not source code

+ Support of incremental approach to system design
  - start with abstract functional representation
  - refinement without breaking structure of model
  - separation of component interface and implementation

+ Valuable feedback from analysis to system designer
  - found several design inconsistencies in satellite case study
  - better understanding of system behavior under (multiple) failures
  - improved safety analysis due to automatic FMEA/FT generation

− Missing (automatic) link between SLIM and engineering models (UML, Simulink)
  - integration into engineering tool suite
Experiences so far

+ Abstraction level of models is appropriate
  ▶ mode transition systems, not source code

+ Support of incremental approach to system design
  ▶ start with abstract functional representation
  ▶ refinement without breaking structure of model
  ▶ separation of component interface and implementation

+ Valuable feedback from analysis to system designer
  ▶ found several design inconsistencies in satellite case study
  ▶ better understanding of system behavior under (multiple) failures
  ▶ improved safety analysis due to automatic FMEA/FT generation

− Missing (automatic) link between SLIM and engineering models (UML, Simulink)
  ▶ integration into engineering tool suite

− Tool set performance on timed and hybrid systems needs further study
Overview

1. Introduction and Challenges
2. System Specification
   - Behavioral Modeling
   - Formal Semantics
   - Error Modeling
   - Property Specification
3. Analysis Facilities
4. Industrial Evaluation
5. Conclusions and Outlook
Epilogue

Achievements:

- Component-based model framework based on AADL
Epilogue

Achievements:

- Component-based model framework based on AADL
- Novelties: hybrid, error modeling, dynamic reconfigurations, ...
Epilogue

Achievements:

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

Achievements:

- Component-based model framework based on AADL
- Novelties: hybrid, error modeling, dynamic reconfigurations, …
- Automated correctness, safety, and performability analysis
- Industrial evaluation by third-party company showed maturity
Epilogue

Achievements:

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

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

Achievements:

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

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

Current and future ESA-funded activities:

▶ Graphical modeling tool (cf. Pierre’s presentation)
▶ FMEA reduction and model slicing
▶ Impact analysis and FDIR synthesis
▶ Compositional model checking
Further Information

▶ Overview paper  
(Yushtein et. al, *IEEE SMC-IT 2011*)

▶ SLIM language  
(Bozzano et. al, *MEMOCODE 2009*)

▶ SLIM formal semantics  
(Bozzano et. al, *Computer J. 2011*)

▶ SLIM model checker  
(Bozzano et. al, *CAV 2010*)

▶ Slicing of SLIM specifications  
(Odenbrett et. al, *NASA FM 2010*)

▶ Tool download at [http://compass.informatik.rwth-aachen.de/](http://compass.informatik.rwth-aachen.de/)