Towards Trustworthy Aerospace Systems: An Experience Report

Joost-Pieter Katoen

Software Modeling and Verification Group
RWTH Aachen University

Invited Talk at Formal Methods in Industrial Critical Systems (FMICS'11)

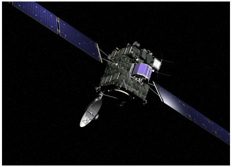
joint work with Marco Bozzano, Alessandro Cimatti, Viet Yen Nguyen, Thomas Noll, Xavier Olivé, Marco Roveri and Yuri Yushstein

RWTH AACHEN UNIVERSITY

Agenda

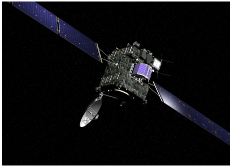
Overview

Aerospace systems



Weather satellite

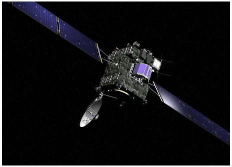
Aerospace systems



Weather satellite

Ariane 5

Aerospace systems

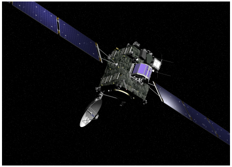


Weather satellite

Ariane 5

Space station ISS

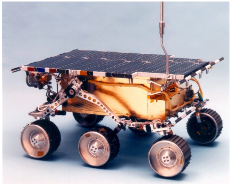
Aerospace systems



Weather satellite

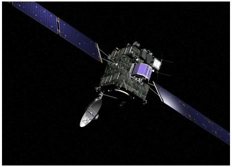
Ariane 5

Space station ISS



Mars Pathfinder

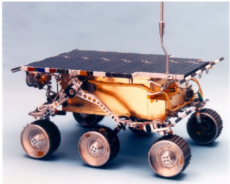
Aerospace systems



Weather satellite

Ariane 5

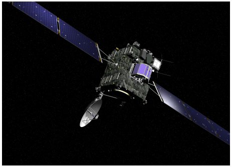
Space station ISS



Mars Pathfinder

GPS system with 26 satellites

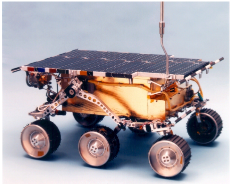
Aerospace systems



Weather satellite

Ariane 5

Space station ISS



Mars Pathfinder

GPS system with 26 satellites

A Lego starwars ship

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 weaknesses and limitations

Software is mostly verified in isolation from the target hardware.

[1] Fault Detection, Identification and Recovery

Current weaknesses and limitations

Software is mostly verified in isolation from the target hardware.

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

[1] Fault Detection, Identification and Recovery

Current weaknesses and limitations

Software is mostly verified in isolation from the target hardware.

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

Distinct modeling formalisms and analysis techniques for different system aspects.

[1] Fault Detection, Identification and Recovery

Current weaknesses and limitations

Software is mostly verified in isolation from the target hardware.

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

Distinct modeling formalisms and analysis techniques for different system aspects.

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

[1] Fault Detection, Identification and Recovery

Current weaknesses and limitations

Software is mostly verified in isolation from the target hardware.

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

Distinct modeling formalisms and analysis techniques for different system aspects.

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

No coherent approach to study effectiveness of FDIR [1]

[1] Fault Detection, Identification and Recovery

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 modelling 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 modelling language, accompanied with a plethora of formal analysis techniques and supported by powerful software tools.

Current situation

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

COMPASS project partners

Consortium

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

Financial support + supervisor

- European Space Agency

Approach in a nutshell

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

Approach in a nutshell

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

Equip this modeling language with a formal semantics.

Approach in a nutshell

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

Equip this modeling language with a formal semantics.

Use specification patterns to ease the specification of system properties.

Approach in a nutshell

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

Equip this modeling language with a formal semantics.

Use specification patterns to ease the specification of system properties.

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

Approach in a nutshell

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

Equip this modeling language with a formal semantics.

Use specification patterns to ease the specification of system properties.

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

Evaluate their effectiveness by industrial case studies.

COMPASS phases

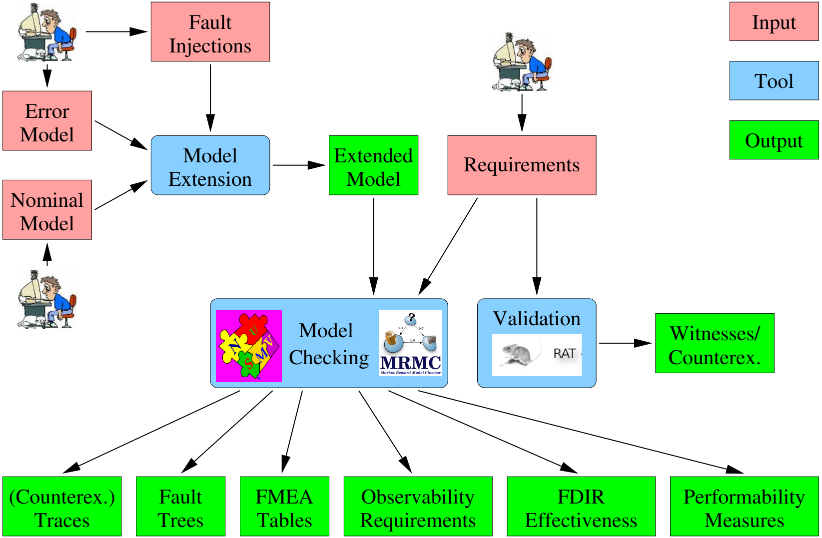
1. Project kick-off — February 2008
2. Language design
3. Software tool specification + software design document
4. Formal semantics — October 2008
5. Prototype tool implementation — April 2009
6. Prototype evaluation
7. Final tool implementation — December 2009
8. Final tool evaluation — March 2010
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

Methodology



Overview

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

The industry standard AADL

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

Paradigm

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

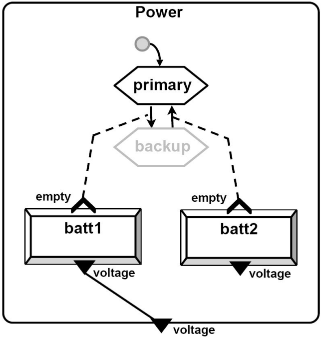
AADL example: redundant power system

Redundant power system

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

We shall show:

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



Modelling a battery in AADL

Component type and implementation:

```
device type Battery


end Battery;
device implementation Battery.Imp




end Battery.Imp;
```

Modelling a battery in AADL

Type defines the interface:

```
device type Battery
  features
    empty: out event port;
    voltage: out data port real initially 6.0;
end Battery;
device implementation Battery.Imp




end Battery.Imp;
```

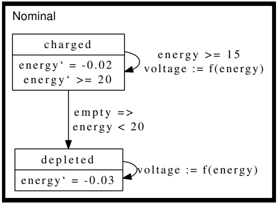
Modelling a battery in AADL

Adding modes behavior:

```
device type Battery
  features
    empty: out event port;
    voltage: out data port real initially 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;
```

Modelling a battery in AADL

Adding hybrid behavior:




```
device type Battery
  features
    empty: out event port;
    voltage: out data port real initially 6.0;
end Battery;
device implementation Battery.Imp
  subcomponents
    energy: data continuous initially 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;
```

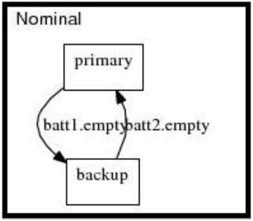
Modeling a redundant power system in AADL

Power system with battery subcomponents:

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

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









end Power.Imp;
```

Modeling a redundant power system in AADL

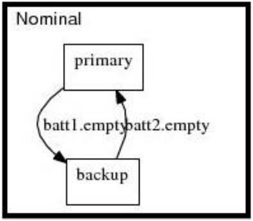
Adding dynamic reconfiguration:




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

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



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

Modeling a redundant power system in AADL

Adding port connections:




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

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

Deviations from AADL

Omissions

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

Deviations from AADL

Omissions

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

Simplifications

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

Deviations from AADL

Omissions

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

Simplifications

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

Extensions

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

Event-data automata

Definition (Event-data automaton)

An event-data automaton (EDA) is a tuple

$$\mathfrak{A} = (M, m_0, X, v_0, \iota, E, \longrightarrow)$$

with

- M finite set of modes
  - $m_0 \in M$ initial mode
- $X = IX \uplus OX \uplus LX$ finite set of input/output/local variables
- $V := \{v \mid v : X \rightarrow \ldots\}$ valuations
  - $v_0 \in V$ initial valuation
- $\iota : M \rightarrow (V \rightarrow \mathbb{B})$ mode invariants (where $\iota(m_0, v_0) = \text{true}$)
- $E = IE \uplus OE$ finite set of input/output events
- $\longrightarrow \subseteq M \times \underbrace{E_\tau}_{\text{trigger}} \times \underbrace{(V \rightarrow \mathbb{B})}_{\text{guard}} \times \underbrace{(V \rightarrow V)}_{\text{effect}} \times M$

  (mode) transition relation (where $E_\tau := E \cup \{\tau\}$)

Semantics of an AADL component

- AADL modes/invariants/transitions
  $\rightsquigarrow$ EDA modes/invariants/transitions

Example (Battery)

- $M = \{\texttt{charged}, \texttt{depleted}\}$, $m_0 = \texttt{charged}$

Semantics of an AADL component

- AADL modes/invariants/transitions
  $\rightsquigarrow$ EDA modes/invariants/transitions
- Incoming/outgoing data ports $\rightsquigarrow$ input/output variables

Example (Battery)

- $M = \{\texttt{charged}, \texttt{depleted}\}$, $m_0 = \texttt{charged}$
- $IX = \varnothing$, $OX = \{\texttt{voltage}\}$

Semantics of an AADL component

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

Example (Battery)

- $M = \{\texttt{charged}, \texttt{depleted}\}$, $m_0 = \texttt{charged}$
- $IX = \varnothing$, $OX = \{\texttt{voltage}\}$
- $LX = \{\texttt{energy}\}$

Semantics of an AADL component

- AADL modes/invariants/transitions
  $\rightsquigarrow$ EDA modes/invariants/transitions
- Incoming/outgoing data ports $\rightsquigarrow$ input/output variables
- Data subcomponents $\rightsquigarrow$ local variables
- Incoming/outgoing event ports $\rightsquigarrow$ input/output events

Example (Battery)

- $M = \{\texttt{charged}, \texttt{depleted}\}$, $m_0 = \texttt{charged}$
- $IX = \emptyset$, $OX = \{\texttt{voltage}\}$
- $LX = \{\texttt{energy}\}$
- $IE = \emptyset$, $OE = \{\texttt{empty}\}$

Operational semantics of EDA

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

Operational semantics of EDA

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

Example (Battery)

$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 100.0, \texttt{voltage} = 6.0 \rangle$$

Operational semantics of EDA

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

Example (Battery)

$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 100.0, \texttt{voltage} = 6.0\rangle$$
$$\downarrow 30.0$$
$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 40.0, \texttt{voltage} = 6.0\rangle$$

Operational semantics of EDA

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

Example (Battery)

$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 100.0, \texttt{voltage} = 6.0\rangle$$
$$\downarrow 30.0$$
$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 40.0, \texttt{voltage} = 6.0\rangle$$
$$\downarrow \tau\langle \texttt{voltage:=...}\rangle$$
$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 40.0, \texttt{voltage} = 4.8\rangle$$

Operational semantics of EDA

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

Example (Battery)

$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 100.0, \texttt{voltage} = 6.0 \rangle$$
$$\downarrow 30.0$$
$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 40.0, \texttt{voltage} = 6.0 \rangle$$
$$\downarrow \tau \langle \texttt{voltage:=...} \rangle$$
$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 40.0, \texttt{voltage} = 4.8 \rangle$$
$$\downarrow 10.0$$
$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 20.0, \texttt{voltage} = 4.8 \rangle$$

Operational semantics of EDA

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

Example (Battery)

$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 100.0, \texttt{voltage} = 6.0 \rangle$$
$$\downarrow 30.0$$
$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 40.0, \texttt{voltage} = 6.0 \rangle$$
$$\downarrow \tau \langle \texttt{voltage:=...} \rangle$$
$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 40.0, \texttt{voltage} = 4.8 \rangle$$
$$\downarrow 10.0$$
$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 20.0, \texttt{voltage} = 4.8 \rangle$$
$$\downarrow \tau \langle \texttt{voltage:=...} \rangle$$
$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 20.0, \texttt{voltage} = 4.4 \rangle$$

Operational semantics of EDA

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

Example (Battery)

$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 100.0, \texttt{voltage} = 6.0 \rangle$$
$$\downarrow 30.0$$
$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 40.0, \texttt{voltage} = 6.0 \rangle$$
$$\downarrow \tau\langle \texttt{voltage:=...} \rangle$$
$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 40.0, \texttt{voltage} = 4.8 \rangle$$
$$\downarrow 10.0$$
$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 20.0, \texttt{voltage} = 4.8 \rangle$$
$$\downarrow \tau\langle \texttt{voltage:=...} \rangle$$
$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 20.0, \texttt{voltage} = 4.4 \rangle$$
$$\downarrow \texttt{empty}$$
$$\langle \texttt{mode} = \texttt{depleted}, \texttt{energy} = 20.0, \texttt{voltage} = 4.4 \rangle$$

Operational semantics of EDA

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

Example (Battery)

$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 100.0, \texttt{voltage} = 6.0 \rangle$$
$$\downarrow 30.0$$
$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 40.0, \texttt{voltage} = 6.0 \rangle$$
$$\downarrow \tau \langle \texttt{voltage:=...} \rangle$$
$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 40.0, \texttt{voltage} = 4.8 \rangle$$
$$\downarrow 10.0$$
$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 20.0, \texttt{voltage} = 4.8 \rangle$$
$$\downarrow \tau \langle \texttt{voltage:=...} \rangle$$
$$\langle \texttt{mode} = \texttt{charged}, \texttt{energy} = 20.0, \texttt{voltage} = 4.4 \rangle$$
$$\downarrow \texttt{empty}$$
$$\langle \texttt{mode} = \texttt{depleted}, \texttt{energy} = 20.0, \texttt{voltage} = 4.4 \rangle$$
$$\downarrow \ldots$$

Networks of event-data automata

Dynamic reconfiguration

$\Longrightarrow$ component activity and port connections mode dependent

Definition (Networks of Event-Data Automata)

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

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

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

- each $\mathfrak{A}_i$ an EDA $\mathfrak{A}_i = (M_i, m_0^i, X_i, v_0^i, \iota_i, E_i, \longrightarrow_i)$
- $M := \prod_{i=1}^{n} M_i$ 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 AADL model

- AADL 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

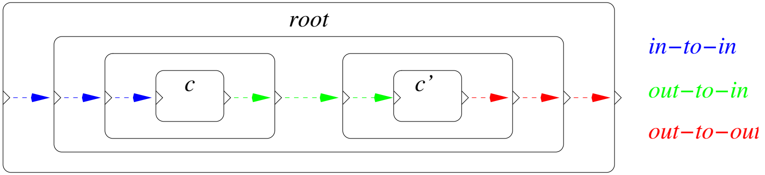
Example (Power System)

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

- $\alpha(\texttt{primary}, m_1, m_2) = \{1, 2\}$
  $\alpha(\texttt{backup}, m_1, m_2) = \{1, 3\}$

Semantics of an entire AADL model

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



Example (Power System)

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

- $EC(\texttt{primary}, m_1, m_2) = \{(2.\texttt{empty}, 1.\texttt{batt1.empty})\}$
  $EC(\texttt{backup}, m_1, m_2) = \{(3.\texttt{empty}, 1.\texttt{batt2.empty})\}$
  $DC(\texttt{primary}, m_1, m_2) = \{(2.\texttt{voltage}, 1.\texttt{voltage})\}$
  $DC(\texttt{backup}, m_1, m_2) = \{(3.\texttt{voltage}, 1.\texttt{voltage})\}$

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 ≥ 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 ≥ 1 connected EDAs
  2. Initialize (re-)activated subcomponents
  3. Establish consistency w.r.t. DC (copy source $\rightarrow$ target data port)

Example (Power system)

$\langle \mathtt{m} = \underline{\mathtt{primary}}, \mathtt{v} = 6.0 \rangle \,|\, \langle \mathtt{m} = \underline{\mathtt{charged}}, \mathtt{e} = 100.0, \mathtt{v} = 6.0 \rangle \,|\, \langle \mathtt{m} = \mathtt{charged}, \mathtt{e} = 100.0, \mathtt{v} = 6.0 \rangle$

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 ≥ 1 connected EDAs
  2. Initialize (re-)activated subcomponents
  3. Establish consistency w.r.t. DC (copy source $\rightarrow$ target data port)

Example (Power system)

$\langle \mathtt{m} = \underline{\mathtt{primary}}, \mathtt{v} = 6.0 \rangle \,|\, \langle \mathtt{m} = \underline{\mathtt{charged}}, \mathtt{e} = 100.0, \mathtt{v} = 6.0 \rangle \,|\, \langle \mathtt{m} = \mathtt{charged}, \mathtt{e} = 100.0, \mathtt{v} = 6.0 \rangle$

$\Downarrow 40.0$

$\langle \mathtt{m} = \underline{\mathtt{primary}}, \mathtt{v} = 6.0 \rangle \,|\, \langle \mathtt{m} = \underline{\mathtt{charged}}, \mathtt{e} = 20.0, \mathtt{v} = 6.0 \rangle \,|\, \langle \mathtt{m} = \mathtt{charged}, \mathtt{e} = 100.0, \mathtt{v} = 6.0 \rangle$

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 ≥ 1 connected EDAs
  2. Initialize (re-)activated subcomponents
  3. Establish consistency w.r.t. *DC* (copy source $\rightarrow$ target data port)

Example (Power system)

$\langle \texttt{m} = \underline{\texttt{primary}}, \texttt{v} = 6.0 \rangle \mid \langle \texttt{m} = \underline{\texttt{charged}}, \texttt{e} = 100.0, \texttt{v} = 6.0 \rangle \mid \langle \texttt{m} = \texttt{charged}, \texttt{e} = 100.0, \texttt{v} = 6.0 \rangle$

$\Downarrow 40.0$

$\langle \texttt{m} = \underline{\texttt{primary}}, \texttt{v} = 6.0 \rangle \mid \langle \texttt{m} = \underline{\texttt{charged}}, \texttt{e} = 20.0, \texttt{v} = 6.0 \rangle \mid \langle \texttt{m} = \texttt{charged}, \texttt{e} = 100.0, \texttt{v} = 6.0 \rangle$

$\Downarrow \tau \langle \texttt{voltage:=...} \rangle$

$\langle \texttt{m} = \underline{\texttt{primary}}, \texttt{v} = 4.4 \rangle \mid \langle \texttt{m} = \underline{\texttt{charged}}, \texttt{e} = 20.0, \texttt{v} = 4.4 \rangle \mid \langle \texttt{m} = \texttt{charged}, \texttt{e} = 100.0, \texttt{v} = 6.0 \rangle$

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 ≥ 1 connected EDAs
  2. Initialize (re-)activated subcomponents
  3. Establish consistency w.r.t. *DC* (copy source → target data port)

Example (Power system)

$$\langle \texttt{m} = \underline{\texttt{primary}}, \texttt{v} = 6.0 \rangle \,|\, \langle \texttt{m} = \underline{\texttt{charged}}, \texttt{e} = 100.0, \texttt{v} = 6.0 \rangle \,|\, \langle \texttt{m} = \underline{\texttt{charged}}, \texttt{e} = 100.0, \texttt{v} = 6.0 \rangle$$
$$\Downarrow 40.0$$
$$\langle \texttt{m} = \underline{\texttt{primary}}, \texttt{v} = 6.0 \rangle \,|\, \langle \texttt{m} = \underline{\texttt{charged}}, \texttt{e} = 20.0, \texttt{v} = 6.0 \rangle \,|\, \langle \texttt{m} = \texttt{charged}, \texttt{e} = 100.0, \texttt{v} = 6.0 \rangle$$
$$\Downarrow \tau\langle \texttt{voltage:=...} \rangle$$
$$\langle \texttt{m} = \underline{\texttt{primary}}, \texttt{v} = 4.4 \rangle \,|\, \langle \texttt{m} = \underline{\texttt{charged}}, \texttt{e} = 20.0, \texttt{v} = 4.4 \rangle \,|\, \langle \texttt{m} = \texttt{charged}, \texttt{e} = 100.0, \texttt{v} = 6.0 \rangle$$
$$\Downarrow \tau\langle \texttt{empty} \rangle$$
$$\langle \texttt{m} = \underline{\texttt{backup}}, \texttt{v} = 6.0 \rangle \,|\, \langle \texttt{m} = \texttt{depleted}, \texttt{e} = 20.0, \texttt{v} = 4.4 \rangle \,|\, \langle \texttt{m} = \underline{\texttt{charged}}, \texttt{e} = 100.0, \texttt{v} = 6.0 \rangle$$

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 ≥ 1 connected EDAs
  2. Initialize (re-)activated subcomponents
  3. Establish consistency w.r.t. *DC* (copy source $\rightarrow$ target data port)

Example (Power system)

$\langle \texttt{m}=\underline{\texttt{primary}}, \texttt{v}=6.0\rangle \mid \langle \texttt{m}=\underline{\texttt{charged}}, \texttt{e}=100.0, \texttt{v}=6.0\rangle \mid \langle \texttt{m}=\texttt{charged}, \texttt{e}=100.0, \texttt{v}=6.0\rangle$

$\Downarrow 40.0$

$\langle \texttt{m}=\underline{\texttt{primary}}, \texttt{v}=6.0\rangle \mid \langle \texttt{m}=\underline{\texttt{charged}}, \texttt{e}=20.0, \texttt{v}=6.0\rangle \mid \langle \texttt{m}=\texttt{charged}, \texttt{e}=100.0, \texttt{v}=6.0\rangle$

$\Downarrow \tau\langle \texttt{voltage:=...}\rangle$

$\langle \texttt{m}=\underline{\texttt{primary}}, \texttt{v}=4.4\rangle \mid \langle \texttt{m}=\underline{\texttt{charged}}, \texttt{e}=20.0, \texttt{v}=4.4\rangle \mid \langle \texttt{m}=\texttt{charged}, \texttt{e}=100.0, \texttt{v}=6.0\rangle$

$\Downarrow \tau\langle \texttt{empty}\rangle$

$\langle \texttt{m}=\underline{\texttt{backup}}, \texttt{v}=6.0\rangle \mid \langle \texttt{m}=\texttt{depleted}, \texttt{e}=20.0, \texttt{v}=4.4\rangle \mid \langle \texttt{m}=\underline{\texttt{charged}}, \texttt{e}=100.0, \texttt{v}=6.0\rangle$

$\Downarrow 40.0$

$\langle \texttt{m}=\underline{\texttt{backup}}, \texttt{v}=6.0\rangle \mid \langle \texttt{m}=\texttt{depleted}, \texttt{e}=20.0, \texttt{v}=4.4\rangle \mid \langle \texttt{m}=\underline{\texttt{charged}}, \texttt{e}=20.0, \texttt{v}=6.0\rangle$

Operational semantics of networks of EDAs

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

Example (Power system)

$$\langle \texttt{m}=\underline{\texttt{primary}}, \texttt{v}=6.0\rangle \,|\, \langle \texttt{m}=\underline{\texttt{charged}}, \texttt{e}=100.0, \texttt{v}=6.0\rangle \,|\, \langle \texttt{m}=\texttt{charged}, \texttt{e}=100.0, \texttt{v}=6.0\rangle$$
$$\Downarrow 40.0$$
$$\langle \texttt{m}=\underline{\texttt{primary}}, \texttt{v}=6.0\rangle \,|\, \langle \texttt{m}=\underline{\texttt{charged}}, \texttt{e}=20.0, \texttt{v}=6.0\rangle \,|\, \langle \texttt{m}=\texttt{charged}, \texttt{e}=100.0, \texttt{v}=6.0\rangle$$
$$\Downarrow \tau\langle \texttt{voltage:=...}\rangle$$
$$\langle \texttt{m}=\underline{\texttt{primary}}, \texttt{v}=4.4\rangle \,|\, \langle \texttt{m}=\underline{\texttt{charged}}, \texttt{e}=20.0, \texttt{v}=4.4\rangle \,|\, \langle \texttt{m}=\texttt{charged}, \texttt{e}=100.0, \texttt{v}=6.0\rangle$$
$$\Downarrow \tau\langle \texttt{empty}\rangle$$
$$\langle \texttt{m}=\underline{\texttt{backup}}, \texttt{v}=6.0\rangle \,|\, \langle \texttt{m}=\texttt{depleted}, \texttt{e}=20.0, \texttt{v}=4.4\rangle \,|\, \langle \texttt{m}=\underline{\texttt{charged}}, \texttt{e}=100.0, \texttt{v}=6.0\rangle$$
$$\Downarrow 40.0$$
$$\langle \texttt{m}=\underline{\texttt{backup}}, \texttt{v}=6.0\rangle \,|\, \langle \texttt{m}=\texttt{depleted}, \texttt{e}=20.0, \texttt{v}=4.4\rangle \,|\, \langle \texttt{m}=\underline{\texttt{charged}}, \texttt{e}=20.0, \texttt{v}=6.0\rangle$$
$$\Downarrow \cdots$$

Error modelling

```
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 modelling

```
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 modelling

```
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 behaviour unless they are linked through fault injection.

Error modelling

```
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

A 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 modelling

```
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 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 = (the original starting mode, the starting error state)
- Set of event ports +:= the 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 = *extended model*

- Modes are pairs of nominal modes and error model states
  - Starting mode = (the original starting mode, the starting error state)
- Set of event ports +:= the 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

Probabilistic error transitions

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

This yields interactive Markov chains := LTS + Markov chains.

Battery component

Nominal specification:

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

end Battery;

device implementation Battery.Imp
  subcomponents
    energy: data continuous initially 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;
```

Battery component after model extension

Product construction for modes:

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

end Battery;

device implementation Battery.Imp
  subcomponents
    energy: data continuous initially 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;
```

Battery component after model extension

Integrate nominal transitions:

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

end Battery;

device implementation Battery.Imp
  subcomponents
    energy: data continuous initially 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;
```

Battery component after model extension

Fault injection:

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

end Battery;

device implementation Battery.Imp
  subcomponents
    energy: data continuous initially 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;
```

Battery component after model extension

Nominal transitions with fault effects:

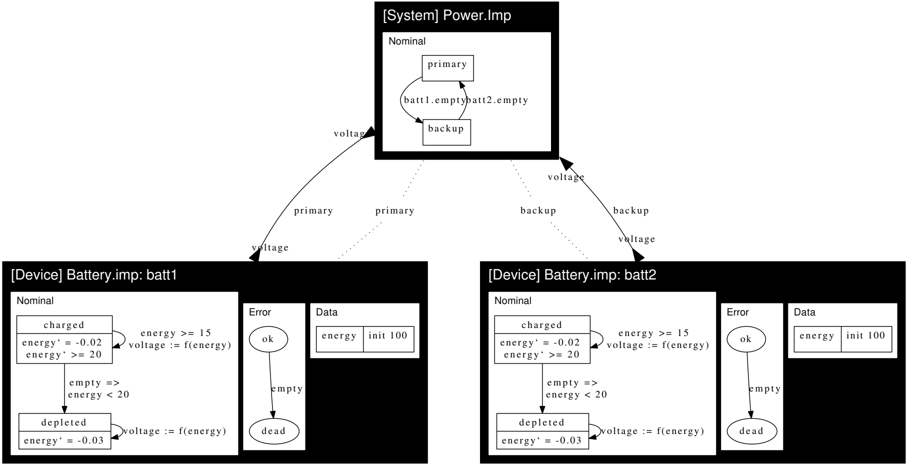
```
device type Battery
  features
    empty: out event port;
    voltage: out data port real initially 6.0;

end Battery;

device implementation Battery.Imp
  subcomponents
    energy: data continuous initially 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;
```

Battery component after model extension

Add error propagations:

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

device implementation Battery.Imp
  subcomponents
    energy: data continuous initially 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;
    depleted#dead -[batteryDied]-> depleted#dead;
```

The complete power system model



Specifying observability

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

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

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

Property specification: Patterns, no formulas!

Examples

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

Property specification: Patterns, no formulas!

Examples

- *The system shall have a behaviour where with probability higher than 0.98 it is the case that voltage* $\geqslant 80$ *holds continously within time bound [0, 10].*
- *The system shall have a behaviour where* $x \leqslant y$ *globally holds.*

Property specification: Patterns, no formulas!

Examples

- $\mathbb{P}_{>0.98}\left(\Box^{[0,10]}(\mathit{voltage} \geqslant 80)\right)$
- $\Box(x \leqslant y)$

Property specification: Patterns, no formulas!

Examples

- $\mathbb{P}_{>0.98}\left(\Box^{[0,10]}(\mathit{voltage} \geqslant 80)\right)$
- $\Box(x \leqslant y)$

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]

Overview

Types of analysis

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 fault modes on events), dynamic FTA
4. Diagnosability
   - FDIR
5. Performance evaluation
   - using probabilistic model checking
   - effective model reduction techniques

Tool components

NuSMV

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

RAT

- Requirements analyser
- Checks logical consistency

FSAP

- Safety analyser
- Fault-tree analysis

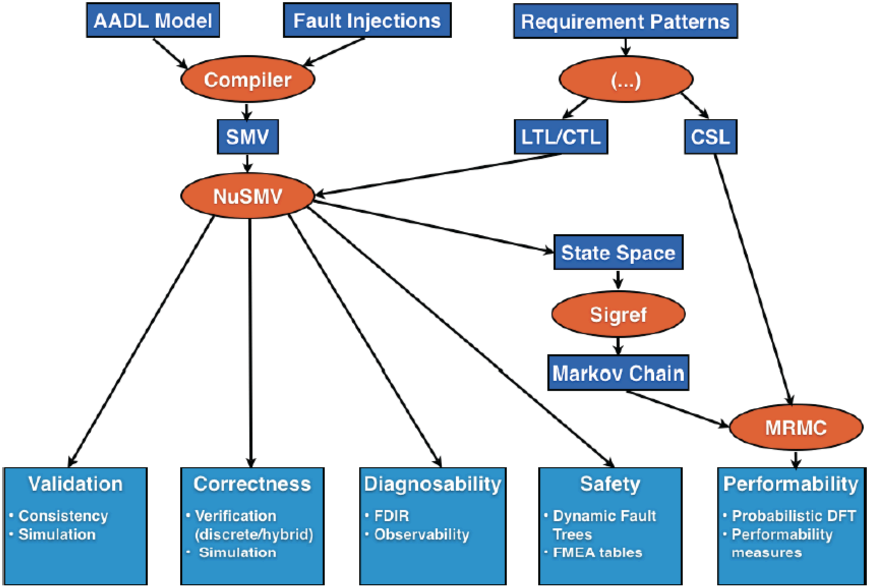
MRMC

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

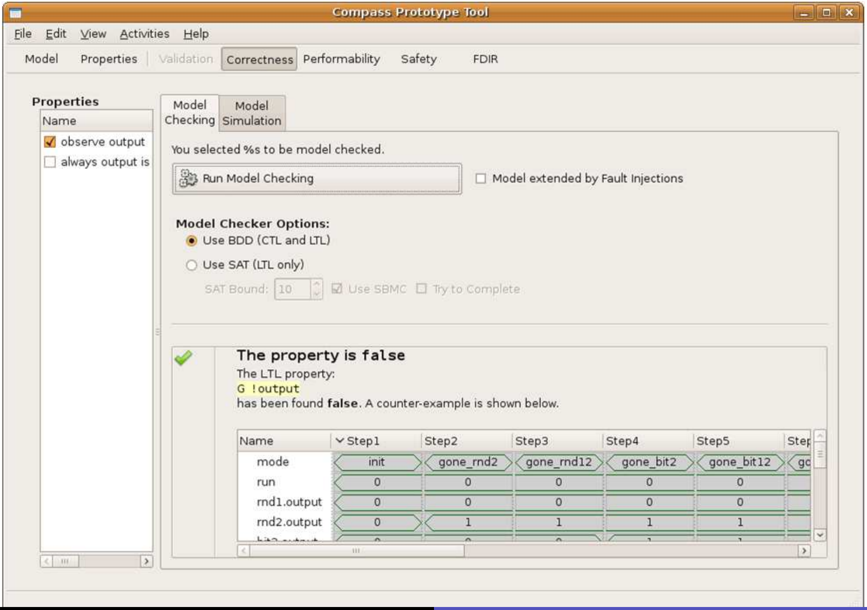
SigRef

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

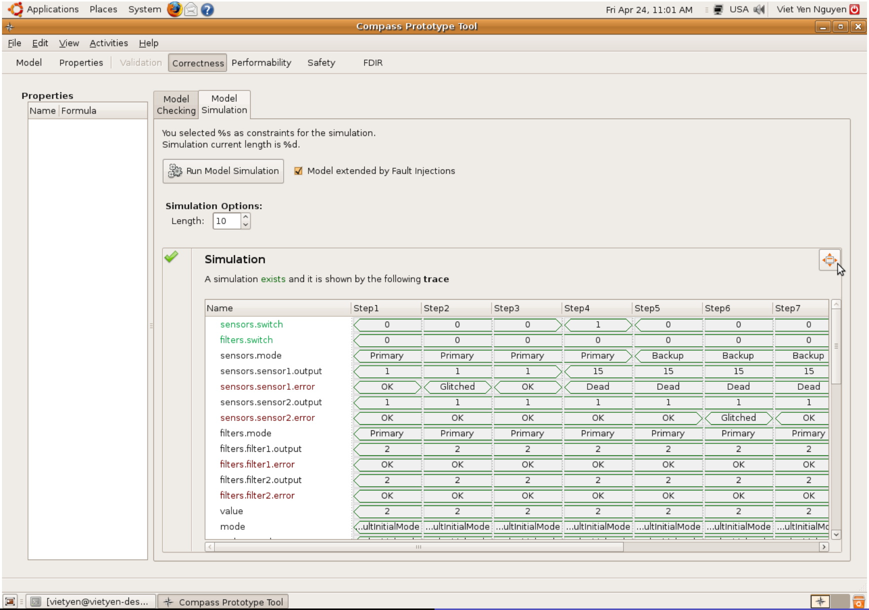
Tool architecture



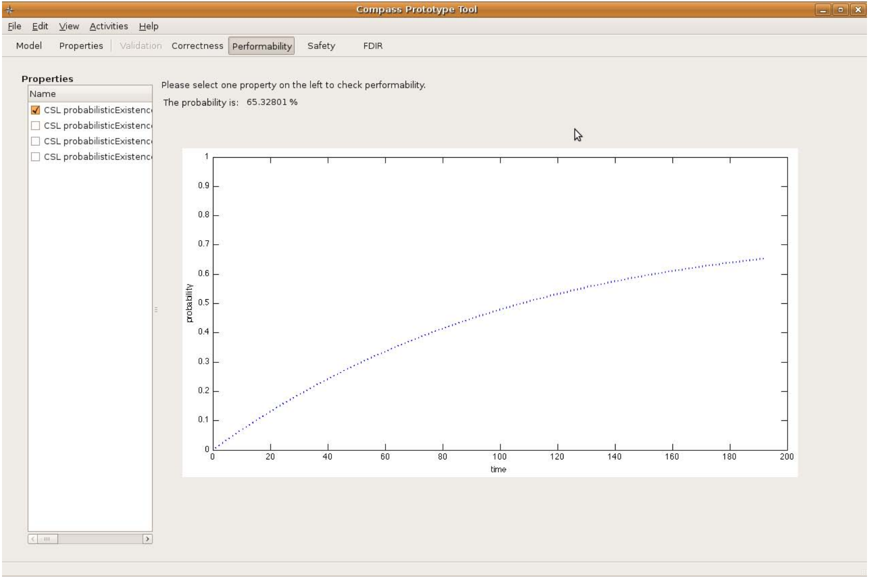
Model checking view



Simulator view

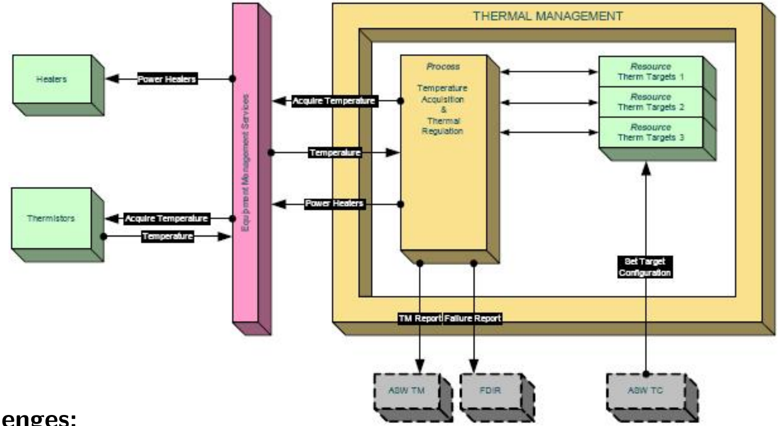


Performance view



Overview

Case study: Satellite thermal regulation



Challenges:

- Hardware (sensors, heaters) and software (control) co-engineering
- Hybrid behavior (temperatures)
- Dynamic reconfiguration (redundancy)
- State-space explosion

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

Case study: Satellite FDIR system

Goal

Assess effectiveness of FDIR measures

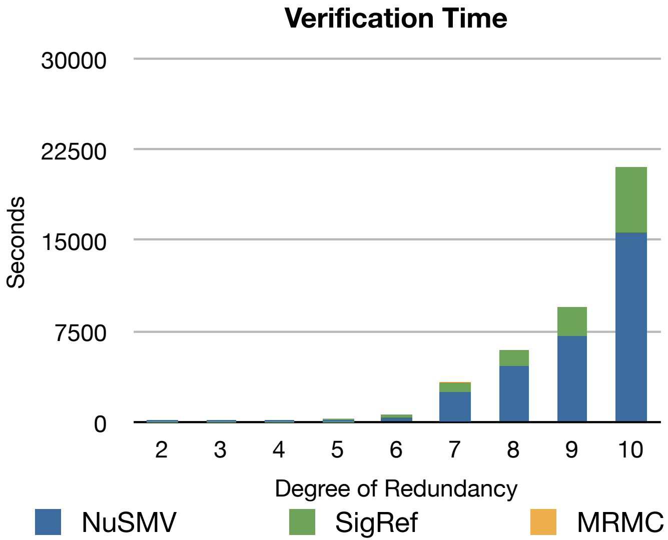
Model components:

- 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

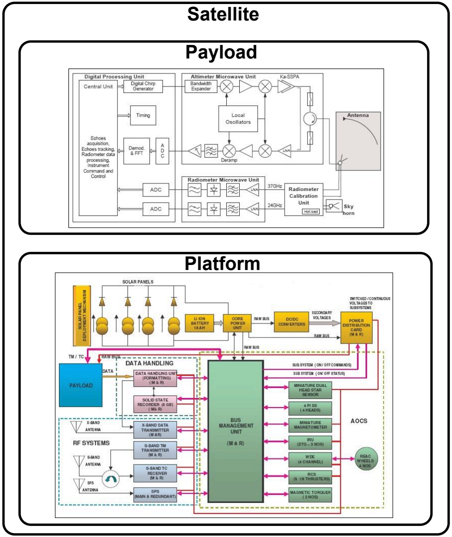


Case study: Satellite of project [redacted]

Launches between 2012-2020

Case study: Satellite of project [redacted]

Launches between 2012-2020

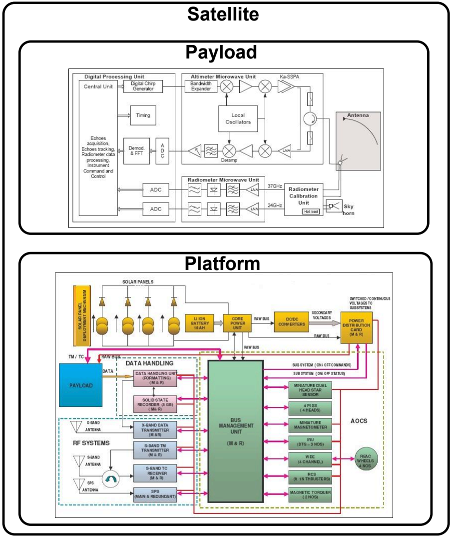


Payload is mission-specific equipment, e.g.:

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

Case study: Satellite of project [redacted]

Launches between 2012-2020



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

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

Platform keeps the satellite orbiting in space, consists of:

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

AADL model of satellite platform

Verification & validation objectives

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

AADL model of satellite platform

Verification & validation objectives

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

Model characteristics

✓ Functional
✓ Probabilistic
✓ Real-time
✓ Hybrid

AADL model of satellite platform

Verification & validation objectives

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

Model characteristics

✓ Functional
✓ Probabilistic
✓ Real-time
✓ Hybrid

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

AADL model of satellite platform

Verification & validation objectives

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

Model characteristics

✓ Functional		Components:	99
✓ Probabilistic		Modes:	217
✓ Real-time		Faults:	21
✓ Hybrid		Recoveries:	9

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

AADL model of satellite platform

Verification & validation objectives

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

Model characteristics

✓ Functional	Components:	99
✓ Probabilistic	Modes:	217
✓ Real-time	Faults:	21
✓ Hybrid	Recoveries:	9

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

Requirement metrics

- Functional properties: 32
- Probabilistic: 2

Analysis results

Analysis results

Correctness and (static) fault tree analysis

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

Analysis	Time
Deadlock checking	173 sec
Model checking "processor alarms only raised on failure"	4 min
Model checking "nominal AOCS thrusters"	102 min
Fault tree analysis "switch to safe modes"	51 min

Analysis results

Correctness and (static) fault tree analysis

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

Analysis	Time
Deadlock checking	173 sec
Model checking "processor alarms only raised on failure"	4 min
Model checking "nominal AOCS thrusters"	102 min
Fault tree analysis "switch to safe modes"	51 min

Setup: AMD Opteron 6172 with 192 GB RAM.

Analysis	Time
Fault detection observables of "processor module"	88 min
Diagnosability of "earth sensors"	$>$ 2.5 days
FMEA table generation	to do
Dynamic fault tree analysis	to do
Performance evaluation	to do

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

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 behaviour under (multiple) failures.
  - Improved safety analysis due to automatic FMEA/FT generation.

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 behaviour under (multiple) failures.
  - Improved safety analysis due to automatic FMEA/FT generation.

– No (automatic) link AADL to engineering models (UML, Simulink)
  - integration into engineering tool suite

– Tool set performance on timed and hybrid systems further study

Overview

Related work

Formal AADL semantics:

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

Related work

Formal AADL semantics:

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

AADL Analysis Tools:

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

Epilogue

Achievements:

Epilogue

Achievements:

- Component-based model framework based on AADL

Epilogue

Achievements:

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

Epilogue

Achievements:

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

Epilogue

Achievements:

- Component-based model framework based on AADL
- Novelties: hybrid, error modelling, 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 modelling, 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

Future and current activities:

- Graphical modelling tool (ESA funded)
- Contribution to AADL standardization
- FMEA reduction, FDIR synthesis, and slicing (ESA funded)
- Compositional model checking (ESA funded)

FMEA reduction

Model	#Classical	#Compact
Thermal regulation (C1)	7	7
Thermal regulation (C2)	67	32
Acquisition (C2)	3	3
Acquisition (C3)	31	4
Command (C1)	1	1
Command (C2)	12	1
Control and Monitoring (C1)	1	1
Control and Monitoring (C2)	12	1
Heating (C1)	1	1
Heating (C2)	13	2
Passive units (C1)	4	4
Passive units (C2)	42	10

Further information

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