Performance Evaluation and Verification of System-Level Architecture Models

Marco Bozzano\textsuperscript{2} Alessandro Cimatti\textsuperscript{2} Marco Roveri\textsuperscript{2}
Joost-Pieter Katoen\textsuperscript{1} Viet Yen Nguyen\textsuperscript{1} Thomas Noll\textsuperscript{1}

\textsuperscript{1}Software Modelling and Verification Group
RWTH Aachen University, Germany

\textsuperscript{2}Embedded Systems Group
Fondazione Bruno Kessler, Italy

University of Oxford, 3 February 2010

\textsuperscript{1}Funded by ESA/ESTEC under Contract No. 21171/07/NL/JD
To show the broad range of applications with current-day model checking techniques, the formal rigor needed to do so, and a glimpse of what industry expects in the future.

1. Case studies
2. AADL
3. Formal Characterisation
4. Injecting Faults
5. COMPASS Toolset
6. Summary and Future Work
Our Case Studies are in the Aerospace Domain

Thales Alenia Space is the leading vendor of satellites in the world, and provides case studies on:

- Global FDIR (Fault Detection, Isolation and Recovery)
- Multiple thruster-based propulsion
- Satellite mode management
- Class 3 thermal regulation management
- Redundant battery discharging and charging
Class 3 Thermal Regulation Management

Requirements

▶ Temperatures shall be acquired within $t$ seconds.
▶ All sensors shall stay between $x$ and $y$ degrees.
▶ Heating interruption delay shall be $s$ minutes in safe mode, and $n$ minutes in nominal mode.
Redundant Battery Discharging and Charging

Requirements

▶ Second battery is always between $x$ and $y$ volts.
▶ Whenever the first battery is empty, the second battery will be activated.
Modelling and Analysing these Case Studies is Difficult

Currently

Plenty of tools and methods for designing and analysing components, but none that deal with overall system behaviour!
Modelling and Analysing these Case Studies is Difficult

Currently

Plenty of tools and methods for designing and analysing components, but none that deal with overall system behaviour!

Pregnant Issues

- Co-engineering: how do hardware and software interact?
- Global FDIR: how do component-local recoveries interact with global system recoveries?
- Degraded modes: what is the system-wide performance?
Modelling and Analysing these Case Studies is Difficult

Currently

Plenty of tools and methods for designing and analysing components, but none that deal with overall system behaviour!

Pregnant Issues

▶ Co-engineering: how do hardware and software interact?
▶ Global FDIR: how do component-local recoveries interact with global system recoveries?
▶ Degraded modes: what is the system-wide performance?

Our Approach

Use the Architecture Analysis and Design Language (AADL), define formal semantics over it, implement an analysis toolset that uses model checking techniques.
Modelling in AADL
AADL: Industry Standard for Modelling Embedded Systems

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

**Features**
- Component-oriented
- HW/SW bindings
- Degraded operations
- Dynamic reconfiguration
- Error handling
- Fault injections
- Timing, probability, hybrid
- Formal semantics
AADL Example: Redundant Power System

We shall show:
- hybrid behaviour of the batteries,
- composition of the power system,
- formalisation to automata,
- semantics as transition systems,
- interweaving of errors.
device type Battery

end Battery;

device implementation Battery.Imp

end Battery.Imp;
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;
AADL: Modelling the Battery
Adding Modes Behaviour

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;
AADL: Modelling the Battery

Adding **Hybrid** Behaviour

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 1.0;
  modes
    charged: activation mode
      while energy’=-0.02 and energy>= 0.2;
    depleted: mode
      while energy’=-0.03 and energy>=0.0;
  transitions
    charged -[then voltage:=2.0*energy+4.0]-> charged;
    charged -[empty when voltage <= 4.4]-> depleted;
    depleted -[then voltage:=2.0*energy+4.0]-> depleted;
end Battery.Imp;
AADL: Modelling the Redundant Power System

Power System with Battery Subcomponents

```plaintext
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;
```
AADL: Modelling the Redundant Power System
Adding Dynamic Reconfiguration

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

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

modes
  primary: initial mode;
  backup: mode;
transitions
  primary -[batt1.empty]-> backup;
  backup -[batt2.empty]-> primary;
end Power.Imp;
```
system 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;
Formal Characterisation
An event-data automaton (EDA) is a tuple

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

with

- \( M \) finite set of modes
- \( m_0 \in M \) initial mode
- \( X = IX \cup OX \cup LX \) 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 \( \iota(m_0, v_0) = \text{true} \))
- \( E = IE \cup OE \) 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 \( E_\tau := E \cup \{\tau\} \))
Formalising AADL Components as Event-Data Automata

- AADL modes/invariants/transitions
  ~ EDA modes/invariants/transitions

Example (Battery)

- \( M = \{\text{charged, depleted}\}, m_0 = \text{charged} \)
Formalising AADL Components as Event-Data Automata

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

Example (Battery)

- $M = \{\text{charged, depleted}\}$, $m_0 = \text{charged}$
- $IX = \emptyset$, $OX = \{\text{voltage}\}$
Formalising AADL Components as Event-Data Automata

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

Example (Battery)

- $M = \{\text{charged, depleted}\}, \ m_0 = \text{charged}$
- $I_X = \emptyset, \ O_X = \{\text{voltage}\}$
- $L_X = \{\text{energy}\}$
Formalising AADL Components as Event-Data Automata

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

Example (Battery)

- \( M = \{ \text{charged, depleted} \} \), \( m_0 = \text{charged} \)
- \( IX = \emptyset, OX = \{ \text{voltage} \} \)
- \( LX = \{ \text{energy} \} \)
- \( IE = \emptyset, OE = \{ \text{empty} \} \)
LTS Semantics of Event-Data Automata

- **States** := $M \times V$
- **Transitions**: timed or internal or event-labeled
LTS Semantics of Event-Data Automata

- States := $M \times V$
- Transitions: timed or internal or event-labeled

Example (Battery)

\[
\langle \text{mode} = \text{charged}, \text{energy} = 1.0, \text{voltage} = 6.0 \rangle
\]
LTS Semantics of Event-Data Automata

- States := \( M \times V \)
- Transitions: timed or internal or event-labeled

**Example (Battery)**

\[
\langle \text{mode} = \text{charged}, \text{energy} = 1.0, \text{voltage} = 6.0 \rangle \\
\downarrow 30.0
\]

\[
\langle \text{mode} = \text{charged}, \text{energy} = 0.4, \text{voltage} = 6.0 \rangle 
\]
LTS Semantics of Event-Data Automata

- States := $M \times V$
- Transitions: timed or internal or event-labeled

Example (Battery)

\[
\langle \text{mode} = \text{charged}, \text{energy} = 1.0, \text{voltage} = 6.0 \rangle \\
\downarrow 30.0
\]

\[
\langle \text{mode} = \text{charged}, \text{energy} = 0.4, \text{voltage} = 6.0 \rangle \\
\downarrow \tau \langle \text{voltage}:=\ldots \rangle \\
\]

\[
\langle \text{mode} = \text{charged}, \text{energy} = 0.4, \text{voltage} = 4.8 \rangle
\]
LTS Semantics of Event-Data Automata

- States := $M \times V$
- Transitions: timed or internal or event-labeled

Example (Battery)

\[
\langle \text{mode} = \text{charged}, \text{energy} = 1.0, \text{voltage} = 6.0 \rangle \\
\downarrow 30.0
\]

\[
\langle \text{mode} = \text{charged}, \text{energy} = 0.4, \text{voltage} = 6.0 \rangle \\
\downarrow \tau \langle \text{voltage} := \ldots \rangle
\]

\[
\langle \text{mode} = \text{charged}, \text{energy} = 0.4, \text{voltage} = 4.8 \rangle \\
\downarrow 10.0
\]

\[
\langle \text{mode} = \text{charged}, \text{energy} = 0.2, \text{voltage} = 4.8 \rangle
\]
LTS Semantics of Event-Data Automata

- States := $M \times V$
- Transitions: timed or internal or event-labeled

**Example (Battery)**

\[
\langle \text{mode} = \text{charged}, \text{energy} = 1.0, \text{voltage} = 6.0 \rangle \\
\downarrow 30.0
\]

\[
\langle \text{mode} = \text{charged}, \text{energy} = 0.4, \text{voltage} = 6.0 \rangle \\
\downarrow \tau \langle \text{voltage} := \ldots \rangle
\]

\[
\langle \text{mode} = \text{charged}, \text{energy} = 0.4, \text{voltage} = 4.8 \rangle \\
\downarrow 10.0
\]

\[
\langle \text{mode} = \text{charged}, \text{energy} = 0.2, \text{voltage} = 4.8 \rangle \\
\downarrow \tau \langle \text{voltage} := \ldots \rangle
\]

\[
\langle \text{mode} = \text{charged}, \text{energy} = 0.2, \text{voltage} = 4.4 \rangle
\]
LTS Semantics of Event-Data Automata

- States := $M \times V$
- Transitions: timed or internal or event-labeled

Example (Battery)

\[
\langle \text{mode} = \text{charged}, \text{energy} = 1.0, \text{voltage} = 6.0 \rangle
\quad \downarrow 30.0
\]

\[
\langle \text{mode} = \text{charged}, \text{energy} = 0.4, \text{voltage} = 6.0 \rangle
\quad \downarrow \tau\langle \text{voltage} = \ldots \rangle
\]

\[
\langle \text{mode} = \text{charged}, \text{energy} = 0.4, \text{voltage} = 4.8 \rangle
\quad \downarrow 10.0
\]

\[
\langle \text{mode} = \text{charged}, \text{energy} = 0.2, \text{voltage} = 4.8 \rangle
\quad \downarrow \tau\langle \text{voltage} = \ldots \rangle
\]

\[
\langle \text{mode} = \text{charged}, \text{energy} = 0.2, \text{voltage} = 4.4 \rangle
\quad \downarrow \text{empty}
\]

\[
\langle \text{mode} = \text{depleted}, \text{energy} = 0.2, \text{voltage} = 4.4 \rangle
\]
LTS Semantics of Event-Data Automata

- States := $M \times V$
- Transitions: timed or internal or event-labeled

Example (Battery)

\[
\begin{align*}
\langle \text{mode} = \text{charged}, \text{energy} = 1.0, \text{voltage} = 6.0 \rangle \\
&\downarrow 30.0 \\
\langle \text{mode} = \text{charged}, \text{energy} = 0.4, \text{voltage} = 6.0 \rangle \\
&\downarrow \tau \langle \text{voltage}:=\ldots \rangle \\
\langle \text{mode} = \text{charged}, \text{energy} = 0.4, \text{voltage} = 4.8 \rangle \\
&\downarrow 10.0 \\
\langle \text{mode} = \text{charged}, \text{energy} = 0.2, \text{voltage} = 4.8 \rangle \\
&\downarrow \tau \langle \text{voltage}:=\ldots \rangle \\
\langle \text{mode} = \text{charged}, \text{energy} = 0.2, \text{voltage} = 4.4 \rangle \\
&\downarrow \text{empty} \\
\langle \text{mode} = \text{depleted}, \text{energy} = 0.2, \text{voltage} = 4.4 \rangle \\
&\downarrow \ldots
\end{align*}
\]
Complete AADL Specifications as Networks of EDAs

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

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

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

- each \( \mathcal{A}_i \) an EDA \( \mathcal{A}_i = (M_i, m_i^0, X_i, v_i^0, \iota_i, E_i, \rightarrow_i) \)
- \( M := \prod_{i=1}^{n} M_i \) set of global modes
- \( \alpha : M \to 2^{[n]} \) activation mapping
- \( EC : M \to (\{i.e \mid i \in [n], e \in E_i\})^2 \) event connection mapping
- \( DC : M \to (\{i.x \mid i \in [n], x \in X_i\})^2 \) data connection mapping
Complete AADL Specifications as Networks of EDAs

- AADL subcomponent in modes declarations
  - activation mapping:
    - root component always active
    - \( c \) active and in mode \( m \), \( sc \) is subcomponent of \( c \), \( sc \) in modes \( m \) 
      \[ \implies sc \text{ active} \]

Example (Power System)

For \( \text{Power/Battery1/Battery2} \):

\[ \begin{align*}
\alpha(\text{primary, charged, charged}) &= \{1, 2\} \\
\alpha(\text{primary, charged, depleted}) &= \{1, 2\} \\
&\quad \ldots \\
\alpha(\text{backup, charged, depleted}) &= \{1, 3\} \\
&\quad \ldots
\end{align*} \]
Complete AADL Specifications as Networks of EDAs

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

Example (Power System)

For $\text{Power/Battery1/Battery2}$:

$EC(\text{primary}, m_1, m_2) = \{(2.\text{empty}, 1.\text{batt1.empty})\}$

$EC(\text{backup}, m_1, m_2) = \{(3.\text{empty}, 1.\text{batt2.empty})\}$

$DC(\text{primary}, m_1, m_2) = \{(2.\text{voltage}, 1.\text{voltage})\}$

$DC(\text{backup}, m_1, m_2) = \{(3.\text{voltage}, 1.\text{voltage})\}$
LTS Semantics of NEDAs

- **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
     - multiway 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)

$\langle m=\text{primary}, v=6.0 \rangle$
$\langle m=\text{charged}, e=1.0, v=6.0 \rangle$
$\langle m=\text{charged}, e=1.0, v=6.0 \rangle$
$\Downarrow$

$40.0$
$\langle m=\text{primary}, v=6.0 \rangle$

$\langle m=\text{charged}, e=0.2, v=6.0 \rangle$
$\langle m=\text{charged}, e=1.0, v=6.0 \rangle$
$\Downarrow$

$\tau$

$\langle \text{voltage:=...} \rangle$
$\langle m=\text{primary}, v=4.4 \rangle$

$\langle m=\text{charged}, e=0.2, v=4.4 \rangle$
$\langle m=\text{charged}, e=1.0, v=6.0 \rangle$
$\Downarrow$

$\tau$

$\langle \text{empty} \rangle$
$\langle m=\text{backup}, v=6.0 \rangle$

$\langle m=\text{depleted}, e=0.2, v=4.4 \rangle$
$\langle m=\text{charged}, e=1.0, v=6.0 \rangle$
$\Downarrow$

$40.0$
$\langle m=\text{backup}, v=6.0 \rangle$

$\langle m=\text{depleted}, e=0.2, v=4.4 \rangle$
$\langle m=\text{charged}, e=0.2, v=6.0 \rangle$
LTS Semantics of NEDAs

- 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
     - multiway 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)

\[
\langle m = \text{primary}, v = 6.0 \rangle \mid \langle m = \text{charged}, e = 1.0, v = 6.0 \rangle \mid \langle m = \text{charged}, e = 1.0, v = 6.0 \rangle
\]
LTS Semantics of NEDAs

- 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
     - multiway 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)

\[
\langle m = \text{primary}, v = 6.0 \rangle | \langle m = \text{charged}, e = 1.0, v = 6.0 \rangle | \langle m = \text{charged}, e = 1.0, v = 6.0 \rangle \\
\downarrow 40.0 \\
\langle m = \text{primary}, v = 6.0 \rangle | \langle m = \text{charged}, e = 0.2, v = 6.0 \rangle | \langle m = \text{charged}, e = 1.0, v = 6.0 \rangle
\]
LTS Semantics of NEDAs

- **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
     - multiway 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)**

\[\langle m = \text{primary}, v = 6.0 \rangle | \langle m = \text{charged}, e = 1.0, v = 6.0 \rangle | \langle m = \text{charged}, e = 1.0, v = 6.0 \rangle \Downarrow 40.0\]

\[\langle m = \text{primary}, v = 6.0 \rangle | \langle m = \text{charged}, e = 0.2, v = 6.0 \rangle | \langle m = \text{charged}, e = 1.0, v = 6.0 \rangle \Downarrow \tau\langle \text{voltage:=...} \rangle\]

\[\langle m = \text{primary}, v = 4.4 \rangle | \langle m = \text{charged}, e = 0.2, v = 4.4 \rangle | \langle m = \text{charged}, e = 1.0, v = 6.0 \rangle\]
LTS Semantics of NEDAs

- **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
     - multiway 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)**

\[\begin{align*}
\langle m=\text{primary}, v=6.0 \rangle & \mid \langle m=\text{charged}, e=1.0, v=6.0 \rangle \mid \langle m=\text{charged}, e=1.0, v=6.0 \rangle \\
& \downarrow 40.0 \\
\langle m=\text{primary}, v=6.0 \rangle & \mid \langle m=\text{charged}, e=0.2, v=6.0 \rangle \mid \langle m=\text{charged}, e=1.0, v=6.0 \rangle \\
& \downarrow \tau \langle \text{voltage:=...} \rangle \\
\langle m=\text{primary}, v=4.4 \rangle & \mid \langle m=\text{charged}, e=0.2, v=4.4 \rangle \mid \langle m=\text{charged}, e=1.0, v=6.0 \rangle \\
& \downarrow \tau \langle \text{empty} \rangle \\
\langle m=\text{backup}, v=6.0 \rangle & \mid \langle m=\text{depleted}, e=0.2, v=4.4 \rangle \mid \langle m=\text{charged}, e=1.0, v=6.0 \rangle
\end{align*}\]
LTS Semantics of NEDAs

- **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
     - multiway event communication from EDA to \(\geq 1\) connected EDAs
  2. Initialize (re-)activated subcomponents
  3. Establish consistency w.r.t. \(DC\) (copy source → target data port)

**Example (Power system)**

\[
\langle m = \text{primary}, v = 6.0 \rangle | \langle m = \text{charged}, e = 1.0, v = 6.0 \rangle | \langle m = \text{charged}, e = 1.0, v = 6.0 \rangle \\
\downarrow 40.0
\]

\[
\langle m = \text{primary}, v = 6.0 \rangle | \langle m = \text{charged}, e = 0.2, v = 6.0 \rangle | \langle m = \text{charged}, e = 1.0, v = 6.0 \rangle \\
\downarrow \tau\langle \text{voltage:=...} \rangle
\]

\[
\langle m = \text{primary}, v = 4.4 \rangle | \langle m = \text{charged}, e = 0.2, v = 4.4 \rangle | \langle m = \text{charged}, e = 1.0, v = 6.0 \rangle \\
\downarrow \tau\langle \text{empty} \rangle
\]

\[
\langle m = \text{backup}, v = 6.0 \rangle | \langle m = \text{depleted}, e = 0.2, v = 4.4 \rangle | \langle m = \text{charged}, e = 1.0, v = 6.0 \rangle \\
\downarrow 40.0
\]

\[
\langle m = \text{backup}, v = 6.0 \rangle | \langle m = \text{depleted}, e = 0.2, v = 4.4 \rangle | \langle m = \text{charged}, e = 0.2, v = 6.0 \rangle
\]
LTS Semantics of NEDAs

- 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
     - multiway 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)

\[
\langle m=\text{primary}, v=6.0 \rangle | \langle m=\text{charged}, e=1.0, v=6.0 \rangle | \langle m=\text{charged}, e=1.0, v=6.0 \rangle \\
\downarrow 40.0
\]
\[
\langle m=\text{primary}, v=6.0 \rangle | \langle m=\text{charged}, e=0.2, v=6.0 \rangle | \langle m=\text{charged}, e=1.0, v=6.0 \rangle \\
\downarrow \tau\{\text{voltage:=...}\}
\]
\[
\langle m=\text{primary}, v=4.4 \rangle | \langle m=\text{charged}, e=0.2, v=4.4 \rangle | \langle m=\text{charged}, e=1.0, v=6.0 \rangle \\
\downarrow \tau\{\text{empty}\}
\]
\[
\langle m=\text{backup}, v=6.0 \rangle | \langle m=\text{depleted}, e=0.2, v=4.4 \rangle | \langle m=\text{charged}, e=1.0, v=6.0 \rangle \\
\downarrow 40.0
\]
\[
\langle m=\text{backup}, v=6.0 \rangle | \langle m=\text{depleted}, e=0.2, v=4.4 \rangle | \langle m=\text{charged}, e=0.2, v=6.0 \rangle \\
\downarrow \ldots
\]
Injecting Faults
Specifying Faulty Behavior

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.
Specifying Faulty Behavior

```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;
```

Fault Injection

An error model does not influence the nominal behaviour unless they are linked through fault injection.
Specifying Faulty Behavior

```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;
```

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.
Specifying Faulty Behavior

```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;
```

Example

In error state `dead`, `voltage:=0`
Model Extension intertwine Faults into the Nominal Model

Nominal model + error model + fault injections = extended model

- Modes are pairs of nominal modes and error model states
  - Starting mode = (original initial mode, initial error state)
- Set of 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 intertwine Faults into the Nominal Model

Nominal model + error model + fault injections = extended model

- Modes are pairs of nominal modes and error model states
  - Starting mode = (original initial mode, initial error state)
- Set of 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

Coping with 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 1.0;
modes
   charged: activation mode while ...;
   depleted: mode while ...;
transitions
   charged -[then voltage:=...]-> charged;
   charged -[empty when voltage <= 4.4]-> depleted;
   depleted -[then voltage:=...]-> depleted;

end Battery.Imp;
Battery Component After **Model Extension**

**Product Construction** for Modes and Error States

```plaintext
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 1.0;
  modes
    charged#ok: activation mode while ...;
    depleted#ok, charged#dead, depleted#dead: mode while ...;
  transitions
    charged -[then voltage:=...] -> charged;
    charged -[empty when voltage <= 4.4] -> depleted;
    depleted -[then voltage:=...] -> depleted;

end Battery.Imp;
```

2010, Viet Yen Nguyen
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 1.0;
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 voltage <= 4.4]-> depleted#ok;
  depleted#ok -[then voltage:=...]-> depleted#ok;

end Battery.Imp;
Battery Component After Model Extension
Add Fault Injections

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 1.0;
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 voltage <= 4.4]-> depleted#ok;
   depleted#ok -[then voltage:=...]-> depleted#ok;
   charged#ok -[then voltage:=0]-> charged#dead;
   depleted#ok -[then voltage:=0]-> depleted#dead;
end Battery.Imp;
Battery Component After **Model Extension**
Nominal Transitions with **Fault Effects**

```plaintext
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 1.0;
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 voltage <= 4.4]-> depleted#ok;
  depleted#ok -[then voltage:=...]-> depleted#ok;
  charged#ok -[then voltage:=0]-> charged#dead;
  depleted#ok -[then voltage:=0]-> depleted#dead;
  charged#dead -[then voltage:=0]-> charged#dead;
  charged#dead -[empty when voltage <= 4.4]-> depleted#dead;
  depleted#dead -[then voltage:=0]-> depleted#dead;
end Battery.Imp;
```

2010, Viet Yen Nguyen
Battery Component After **Model Extension**

**Add Error Propagations**

```plaintext
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 1.0;
  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 voltage <= 4.4]-> depleted#ok;
    depleted#ok -[then voltage:=...]-> depleted#ok;
    charged#ok -[then voltage:=0]-> charged#dead;
    depleted#ok -[then voltage:=0]-> depleted#dead;
    charged#dead -[then voltage:=0]-> charged#dead;
    charged#dead -[empty when voltage <= 4.4]-> depleted#dead;
    depleted#dead -[then voltage:=0]-> depleted#dead;
    depleted#dead -[batteryDied]-> depleted#dead;
    charged#dead -[batteryDied]-> charged#dead;
end Battery.Imp;
```

2010, Viet Yen Nguyen
Analysing AADL Models
SigRef applies weak stochastic bisimulation minimisation to get a smaller IMC.

If the IMC is deterministic, it is a Continuous Time Markov Chain, for which stochastic model checking exist.
Scalability of Performability Evaluation

Verification Time

Seconds

Verification Time

Degree of Redundancy

NuSMV

SigRef

MRMC

Observation

NuSMV model checker contributes most to the overall runtime.
Summary

Comprehensive approach for formal analysis of (critical) embedded system architectures
(like aerospace systems)

To this aim, we developed:

- Formal semantics of AADL and its Error Annex in terms of (Network of) Event-Data Automata.
  - Defined analyses for correctness, performance, dependability and RAMS aspects over AADL.
  - Developed graphical tool support for it.
- AADL Standards Body plans to incorporate our extensions.
- Defined a methodological integration into ECSS.
- Case studies to evaluate the approach.
Selected References

By Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll and Marco Roveri:


▶ Formal Verification and Validation of AADL Models in proceedings of ERTSS, 2010


Slides of COMPASS 2009 workshop talks at ETAPS’09 are available:

compass.informatik.rwth-aachen.de
Future Work

Usual...
Handling bigger models, improving usability, . . .

Mixing synchronous and asynchronous behaviour
Our AADL semantics describe asynchronous behaviour. For some case studies, synchronous behaviour, and - if possible - a mix of those two have been requested.

Improved diagnosability
Currently, the toolset can compute whether a diagnosis can be performed or not given a fixed set of observables. However, the ultimate aim is to cut down on the amount of observables (i.e., sensors) needed to derive a proper diagnosis.

Increasing flexibility of error modelling
Currently, error behaviour is described per component instance. We got feedback from Thales that they also would like to model error behaviour per nominal mode.