TRUSTWORTHY DESIGN
VALIDATION OF A
SATELLITE PLATFORM

Viet Yen Nguyen
nguyen@cs.rwth-aachen.de

this research is commissioned by:

MDDays 2013, Eindhoven, Netherlands
OVERVIEW

1. Space System Software Design & Validation
2. Formal methods with COMPASS toolset
3. Satellite case study
4. Conclusions
TECHNOLOGY CONTEXT OF THIS TALK

- Space Engineering
  - Industry
    - Control
    - Requirements
    - Systems
    - Software
    - Hardware
    - Thermal
    - Safety & Dependability
    - Cost
  - Thermal
  -... 

- Formal Methods
  - Academia
    - Formal Semantics
    - Logics
    - Model Checking
    - SAT-solving
    - Reachability
    - Markov Chains
    - Small-Step Semantics
    - Timed automata
    - Hybrid systems
    -... 

-...
CASE: SATELLITE OF ESA MISSION IN DEVELOPMENT

- Platform keeps satellite in space, like car’s chassis.
  - control & data unit,
  - propulsion,
  - telemetry, tracking & cmd.
  - power,
  - attitude & orbit control sys.,
  - reconfiguration module,
  - etc.

- Fault detection, isolation, recovery (FDIR) software and hardware:
  - redundancies + recovery,
  - compensation algorithms,
  - failure isolation schemes,
  - omnipresent in satellite.

Note: shown satellite is not the one of the case study.
DEMANDING REQUIREMENTS NEED EXTENSIVE V&V

- “Satellite’s in-orbit lifetime shall be at least 5 years.”
- “Satellite’s platform reliability shall be equivalent or less than 1000 FIT.”
- “The probability that the satellite transits to safe mode shall be lower than 0.005, considering 40 hours for ground time to repair.”
- “In case of an over-current, a switch to the redundant coil is performed by OBDH SW.”
- “While in nominal mode and an lost uplink, the transponder shall transit within 10 seconds to safe mode.”

- … thousands of similar requirements!

- In presence of space hazards like radiation, extreme heat/cold, radio latencies, gravitational forces, etc.

Note: shown requirements are not from the case study.
### EXAMPLE OF V&V IN SPACE SYSTEMS ENGINEERING

**Table:**

<table>
<thead>
<tr>
<th>Activities</th>
<th>Phases</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Phase 0</td>
</tr>
<tr>
<td>Mission/Function</td>
<td></td>
</tr>
<tr>
<td>Requirements</td>
<td></td>
</tr>
<tr>
<td>Definition</td>
<td></td>
</tr>
<tr>
<td>Verification</td>
<td></td>
</tr>
<tr>
<td>Production</td>
<td></td>
</tr>
<tr>
<td>Utilization</td>
<td></td>
</tr>
<tr>
<td>Disposal</td>
<td></td>
</tr>
</tbody>
</table>

**Activities:***

- **Requirements analysis.**
- **Fault tree analysis.**
- **Review and inspection.**
- **HW/SW co-testing.**
- **Early prototyping.**
- **Control simulation.**
- **In orbit testing.**
- **End-to-end IS testing.**
- **Operations readiness.**
- **Mission scenario tests.**
SPACECRAFT IS BECOMING “FLYING SOFTWARE”

Growth in Code Size for Human and Robotic Missions

From NASA Study Flight Software Complexity (2009). Same growth trend in:
- military aircrafts,
- cars, and
- airplanes.
CHALLENGES IN SPACE VERIFICATION & VALIDATION

- More demanding requirements.
- More FDIR software.
- More system behaviors.
- More issues during V&V.
- More pressure on meeting cost and schedule constraints!

Example: once every 175 year launch window for Voyager 2.
OUR SOLUTION: COMPASS

Formal-methods based system software co-engineering
HISTORY: COMPASS CONSORTIUM

Formal semantics, performance evaluation, model checking, etc.

Model-based engineering tools

Domain experts

Model checking, safety & dependability analysis, etc.
### INCREASE FORMALITY IN EARLY PHASE B

#### Activities vs. Phases

<table>
<thead>
<tr>
<th>Activities</th>
<th>Phase 0</th>
<th>Phase A</th>
<th>Phase B</th>
<th>Phase C</th>
<th>Phase D</th>
<th>Phase E</th>
<th>Phase F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mission/Function</td>
<td></td>
<td></td>
<td>MDR</td>
<td>PRR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements</td>
<td></td>
<td></td>
<td>SRR</td>
<td>PDR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Definition</td>
<td></td>
<td></td>
<td></td>
<td>CDR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verification</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>QR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Production</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AR</td>
<td>FRR</td>
</tr>
<tr>
<td>Utilization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CRR</td>
<td>ELR</td>
</tr>
<tr>
<td>Disposal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LRR</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MCR</td>
</tr>
</tbody>
</table>

**Improve understanding of system software behavior with early V&V.**

1. Model-based.
2. Formal methods.
3. Automated analysis tools.
MODEL-BASED ANALYSIS FOR V&V USING COMPASS

- **Model**
  - Nominal
  - Error

- **Requirements**
  - Property Patterns
    - CTL/LTL
    - CSL

- **Automated Analyses**
  - Simulation
    - State exploration
  - Model Checking
    - Reachability
  - Fault Tree Analysis
  - FMEA
    - Reachability
  - Probabilistic Risk Assessment
    - Probabilistic transient
  - Diagnosis Constraints
    - TwinPlant reachability
ARCHITECTURE & ANALYSIS DESIGN LANGUAGE:
ONE LANGUAGE FOR ALL SYSTEM-LEVEL ASPECTS

- Originated on MetaH from 1989
- Standard by SAE (Society of Automotive Engineers)
- Models real-time and performance critical HW/SW architecture
- Hierarchical and component-based.
NOMINAL MODELING

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

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

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

error model implementation BatteryFailure.Imp
  events
    fault: error event occurrence poisson 0.001;
    works: error event occurrence poisson 0.2;
    fails: error event occurrence poisson 0.8;
  transitions
    ok -[fault]-> dead;
    dead -[batteryDied]-> dead;
    dead -[reset]-> resetting;
    resetting -[works]-> ok;
    resetting -[fails]-> dead;
end BatteryFailure.Imp;
FAULT INJECTION AND MODEL EXTENSION

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

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

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

error model implementation BatteryFailure.Imp
  events
    fault: error event occurrence poisson 0.001;
    works: error event occurrence poisson 0.2;
    fails: error event occurrence poisson 0.8;
  transitions
    ok -[fault]-> dead;
    dead -[batteryDied]-> dead;
    dead -[reset]-> resetting;
    resetting -[works]-> ok;
    resetting -[fails]-> dead;
  end BatteryFailure.Imp;

Dead: voltage := 0

Nominal

Erroneous

Fault injection

Extended model
Operational semantics:

- System is a tuple of variables.
  \( (\text{mode}, \text{alert}, \text{battery1.mode}, \ldots) \)

- System state is a valuation to tuple variables.
  \(<\text{primary}, \text{false}, \text{charged}, \ldots>\>

- Nominal/error transitions change the current state to a new system state.
  \(<\text{primary}, \text{false}, \text{charged}, \ldots> \rightarrow \emptyset \rightarrow <\text{backup}, \text{false}, \text{charged}, \ldots>\>

Fundamental to formal methods.

- Precise & unambiguous
- Automated analyses

\begin{align*}
\langle \text{primary}, [\text{alert} \rightarrow \bot] \rangle & || \\
\langle \text{charged}, [\text{energy} \rightarrow 1.0, \text{tryReset} \rightarrow \bot, \text{voltage} \rightarrow 6.0] \rangle & || \\
\langle \text{charged}, [\text{energy} \rightarrow 1.0, \text{tryReset} \rightarrow \bot, \text{voltage} \rightarrow 6.0] \rangle & || \\
\langle m_0, [\text{voltage} \rightarrow 6.0, \text{alert} \rightarrow \bot] \rangle & \downarrow 30.0 \\
\langle \text{primary}, [\text{alert} \rightarrow \bot] \rangle & || \\
\langle \text{charged}, [\text{energy} \rightarrow 0.4, \text{tryReset} \rightarrow \bot, \text{voltage} \rightarrow 6.0] \rangle & || \\
\langle \text{charged}, [\text{energy} \rightarrow 1.0, \text{tryReset} \rightarrow \bot, \text{voltage} \rightarrow 6.0] \rangle & || \\
\langle m_0, [\text{voltage} \rightarrow 4.8, \text{alert} \rightarrow \bot] \rangle & \downarrow \tau \\
\langle \text{primary}, [\text{alert} \rightarrow \bot] \rangle & || \\
\langle \text{charged}, [\text{energy} \rightarrow 0.2, \text{tryReset} \rightarrow \bot, \text{voltage} \rightarrow 4.8] \rangle & || \\
\langle \text{charged}, [\text{energy} \rightarrow 1.0, \text{tryReset} \rightarrow \bot, \text{voltage} \rightarrow 6.0] \rangle & || \\
\langle m_0, [\text{voltage} \rightarrow 4.4, \text{alert} \rightarrow \bot] \rangle & \downarrow \tau \\
\langle \text{primary}, [\text{alert} \rightarrow \top] \rangle & || \\
\langle \text{charged}, [\text{energy} \rightarrow 0.2, \text{tryReset} \rightarrow \top, \text{voltage} \rightarrow 4.4] \rangle & || \\
\langle \text{charged}, [\text{energy} \rightarrow 1.0, \text{tryReset} \rightarrow \bot, \text{voltage} \rightarrow 6.0] \rangle & || \\
\langle m_0, [\text{voltage} \rightarrow 4.4, \text{alert} \rightarrow \top] \rangle & \downarrow \emptyset \\
\langle \text{primary}, [\text{alert} \rightarrow \bot] \rangle & || \\
\langle \text{depleted}, [\text{energy} \rightarrow 0.2, \text{tryReset} \rightarrow \top, \text{voltage} \rightarrow 4.4] \rangle & || \\
\langle \text{charged}, [\rightarrow 1.0, \text{tryReset} \rightarrow \bot, \text{voltage} \rightarrow 6.0] \rangle & || \\
\langle m_0, [\text{voltage} \rightarrow 6.0, \text{alert} \rightarrow \bot] \rangle & \downarrow : 
\end{align*}
### Patterns

- The system shall have a behavior where \( 80 \leq \text{voltage} \leq 90 \) 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

- \( \Box (80 \leq \text{voltage} \leq 90) \) (Linear Temporal Logic)
- \( P > 0.98 (\Box [0, 10] (\text{voltage} \geq 80)) \) (Continuous Stochastic Logic)
Case Study Model

Detailed version of this is confidential.

Case Study Model
### Six months. Parallel with satellite development:
1. Nominal design architecture matured.
2. FDIR architecture volatile and in development.
3. Many details were yet to be decided.
CASE: SATELLITE OF ESA MISSION IN DEVELOPMENT

- Platform keeps satellite in space, like car’s chassis.
  - control & data unit,
  - propulsion,
  - telemetry, tracking & cmd.
  - power,
  - attitude & orbit control sys.,
  - reconfiguration module,
  - ...

- Fault detection, isolation, recovery (FDIR) software and hardware:
  - redundancies + recovery,
  - compensation algorithms,
  - failure isolation schemes,
  - omnipresent in satellite.

Note: shown satellite is not the one of the case study.
DESCRIPTION: WHAT’S MODELLED - COVERAGE

Most particularly:

- Voting algorithms on hot redundant elements.
- Off-range checkers.
- Initialization sequences.
- Recovery sequences, disabling/enabling system elements.
- Flags:
  - Status
  - Commandability
- Failure configurations.
- Evolution of physical quantities:
  - Time
  - Pressure
  - Heat
- Timing constraints.
## Largest System-Level Model Ever Developed

<table>
<thead>
<tr>
<th>Scope</th>
<th>Metric</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model</strong></td>
<td>Components</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>Ports</td>
<td>937</td>
</tr>
<tr>
<td></td>
<td>Modes</td>
<td>244</td>
</tr>
<tr>
<td></td>
<td>Error models</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Recoveries</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>Nominal state space</td>
<td>48421100</td>
</tr>
<tr>
<td></td>
<td>LOC (without comments)</td>
<td>3831</td>
</tr>
<tr>
<td><strong>Requirements</strong></td>
<td>Propositional</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Absence</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Universality</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Response</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>Probabilistic Invariance</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Probabilistic Existence</td>
<td>1</td>
</tr>
</tbody>
</table>
All analysis results are in the publication.

CASE STUDY ANALYSES
INJECTING FAULTS EXPLODE THE STATE SPACE

- Single earth sensor signal failure: Multiplication of state space - 3, Number of injections - 1
- Double earth sensors signal failure: Multiplication of state space - 11, Number of injections - 2
- Propulsion failure: Multiplication of state space - 35, Number of injections - 5
- AOCS equipments failure: Multiplication of state space - 213563, Number of injections - 7
- All reactionwheel failures: Multiplication of state space - 1372, Number of injections - 16
- Single reactionwheel failure: Multiplication of state space - 10, Number of injections - 10
- Processor module failures: Multiplication of state space - 1372, Number of injections - 10
- Complete earth sensor failure: Multiplication of state space - 172, Number of injections - 11
- Reactionwheel + earth sensor failures: Multiplication of state space - 172, Number of injections - 22

Explosion correlates with scope of failure.
Adding requirements

Loading SLIM models

Adding fault injections

Loading SLIM models

Adding requirements

Adding fault injections
COMPASS TOOLSET: SIMULATION

Adding fault injections

Choose a transition

Simulation states

Simulation

BDD model checking of CTL formula for discrete case

SMT model checking of LTL formula for hybrid case

Counterexample when property does not hold

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

Fault tree contains Priority AND, AND, OR-gates and edges represent triggers.

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

Fault tree to interactive Markov chain. Minimise to continuous-time Markov chain. Probabilistic model check it.

Determine effects on nominal model upon combinations of failures.

"what happens when sensor dies?"

Determine system performance while under degraded modes.

Transforms full state space to a Markov chain. Probabilistic model check it.

Determine whether sufficient alarms are available to deduce faults.

Which alarms are triggered on failure?

Does the system recover from a failure?
Adding fault injections

Simulation states

Verify a property e.g. “voting scheme on sensor values always compensate double sensor failures”

BDD model checking of CTL formula for discrete case

SMT model checking of LTL formula for hybrid case

Counterexample when property does not hold

Relate faults to higher-level failures e.g. “which faults lead to undervoltage?”

Fault tree contains Priority AND, AND, OR-gates and edges represent triggers.

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

Fault tree to interactive Markov chain. Minimise to continuous-time Markov chain. Probabilistic model check it.

Determine effects on nominal model upon combinations of failures.

“What happens when sensor dies?”

Determine system performance while under degraded modes.

Transforms full state space to a Markov chain. Probabilistic model check it.

Determine whether sufficient alarms are available to deduce faults.

Which alarms are triggered on failure?

Does the system recover from a failure?
COMPASS TOOLSET: FAULT TREE ANALYSIS

- Adding fault injections
- Simulation states
- BDD model checking of CTL formula for discrete case
- SMT model checking of LTL formula for hybrid case
- Counterexample when property does not hold

Relate faults to higher-level failures e.g. “which faults lead to undervoltage?”

- Reachability analysis with history variables
- Fault tree contains Priority AND, AND, OR-gates and edges represent triggers.

Fault tree to interactive Markov chain. Minimise to continuous-time Markov chain. Probabilistic model check it.

Determine effects on nominal model upon combinations of failures.

“what happens when sensor dies?”

Determine system performance while under degraded modes.

Transforms full state space to a Markov chain. Probabilistic model check it.

Determine whether sufficient alarms are available to deduce faults.

Which alarms are triggered on failure?

Does the system recover from a failure?
Adding fault injections
Simulation states
BDD model checking of CTL formula for discrete case
SMT model checking of LTL formula for hybrid case
Counterexample when property does not hold
Relate faults to higher-level failures e.g. "which faults lead to undervoltage?"
Fault tree contains Priority AND, AND, OR-gates and edges represent triggers.
Computes probability of top-level event (e.g. system failure)
Fault tree to interactive Markov chain. Minimise to continuous-time Markov chain. Probabilistic model check it.
Determine effects on nominal model upon combinations of failures.
"what happens when sensor dies?"
Determine system performance while under degraded modes.
Transforms full state space to a Markov chain. Probabilistic model check it.
Determine whether sufficient alarms are available to deduce faults.
Which alarms are triggered on failure?
Does the system recover from a failure?
Adding fault injections

Simulation states
BDD model checking of CTL formula for discrete case
SMT model checking of LTL formula for hybrid case

Counterexample when property does not hold
Relate faults to higher-level failures e.g. “which faults lead to undervoltage?”

Fault tree contains Priority AND, AND, OR-gates and edges represent triggers.
Computes probability of top-level event (e.g. system failure)
Fault tree to interactive Markov chain. Minimise to continuous-time Markov chain. Probabilistic model check it.
Determine effects on nominal model upon combinations of failures.

“what happens when sensor dies?”

“value range out of range!”
Adding fault injections

Simulation states

BDD model checking of CTL formula for discrete case

SMT model checking of LTL formula for hybrid case

Counterexample when property does not hold

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

Fault tree contains Priority AND, AND, OR-gates and edges represent triggers.

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

Fault tree to interactive Markov chain. Minimise to continuous-time Markov chain. Probabilistic model check it.

Determine effects on nominal model upon combinations of failures.

"what happens when sensor dies?"

Determine system performance while under degraded modes.

Transforms full state space to a Markov chain. Probabilistic model check it.

Determine whether sufficient alarms are available to deduce faults.

Which alarms are triggered on failure?

Does the system recover from a failure?
Determine whether sufficient alarms are available to deduce faults.
COMPASS TOOLSET: FDIR EFFECTIVENESS

- Which alarms are triggered on failure?
- Which failures trigger an alarm?
- Does the system recover from a failure?
EPILOGUE
FACT SHEET OF THE COMPASS TOOLSET

• Reused core software
  NuSMV [Cim+02], MRMC [Kat+11], Sigref [Wim+06]

• Subcontractors
  FBK (Italy), Thales (France), Ellidiss (France)

• Budget
  $\approx 750,000$ Euro

• Programmers
  $\approx 12$ persons

• Lines of code
  $\approx 100,000$ lines of Python

• Development time
  $\approx 3$ years

• Reported Trac issues
  $\geq 500$

• Used languages
  Python, C, C++

• Licenses
  • COMPASS License
    distribution in ESA member-states only
  • GNU Public License
    open source
  • FBK License
    binary-only non-distributable
  • CGM License
    binary-only non-distributable non-commercial-use
  • US transfer license
    for use by a specific US-based organisation

• Website
  http://compass.informatik.rwth-aachen.de/
BEYOND TODAY: ESA AND EU FUND FOLLOW-UPS

- **ESA projects**
  - AUTOGEF  automated synthesis of FDIR logic
  - FAME    methodology for FDIR development lifecycle
  - HASDEL  enhancements for launcher/rocket development

- **EU projects**
  - D-MILS  security aspects in system-level architectures

These project’s budgets account for over 4 million EUR.
CONCLUSION

State-of-the-art of validating spacecraft software designs for correctness, safety, dependability and performance aspects using a single system model.

• Modeling language
  – Based on AADL industry standard
  – Graphical and textual
  – Formal semantics
    • Discrete
    • Real-time
    • Hybrid
    • Probabilistic

• Extensive validation approach
  – Correctness
  – Safety & dependability
  – Performability
  – FDIR effectiveness

• Engineer-friendly toolset
  – State of the art (probabilistic) model checking.

• Spacecraft case studies
  – Satellite platforms
    • Phase B
    • Phase C
  – Satellite subsystems
    • FDIR mode management
    • Thermal regulation system

• Follow-up projects
  – Synthesis of FDIR logic
  – Methodology and development lifecycle of FDIR systems
  – Launcher/rocket development
  – Security architectures
MORE INFORMATION


- COMPASS Toolset: compass.informatik.rwth-aachen.de
  - Freely available for everyone in ESA-member states.