A case study in formal system engineering with SysML

Iulia Dragomir[1], Iulian Ober[1] and David Lesens[2]

[1]IRIT - University of Toulouse

[2]Astrium Space Transportation

July 19, 2012

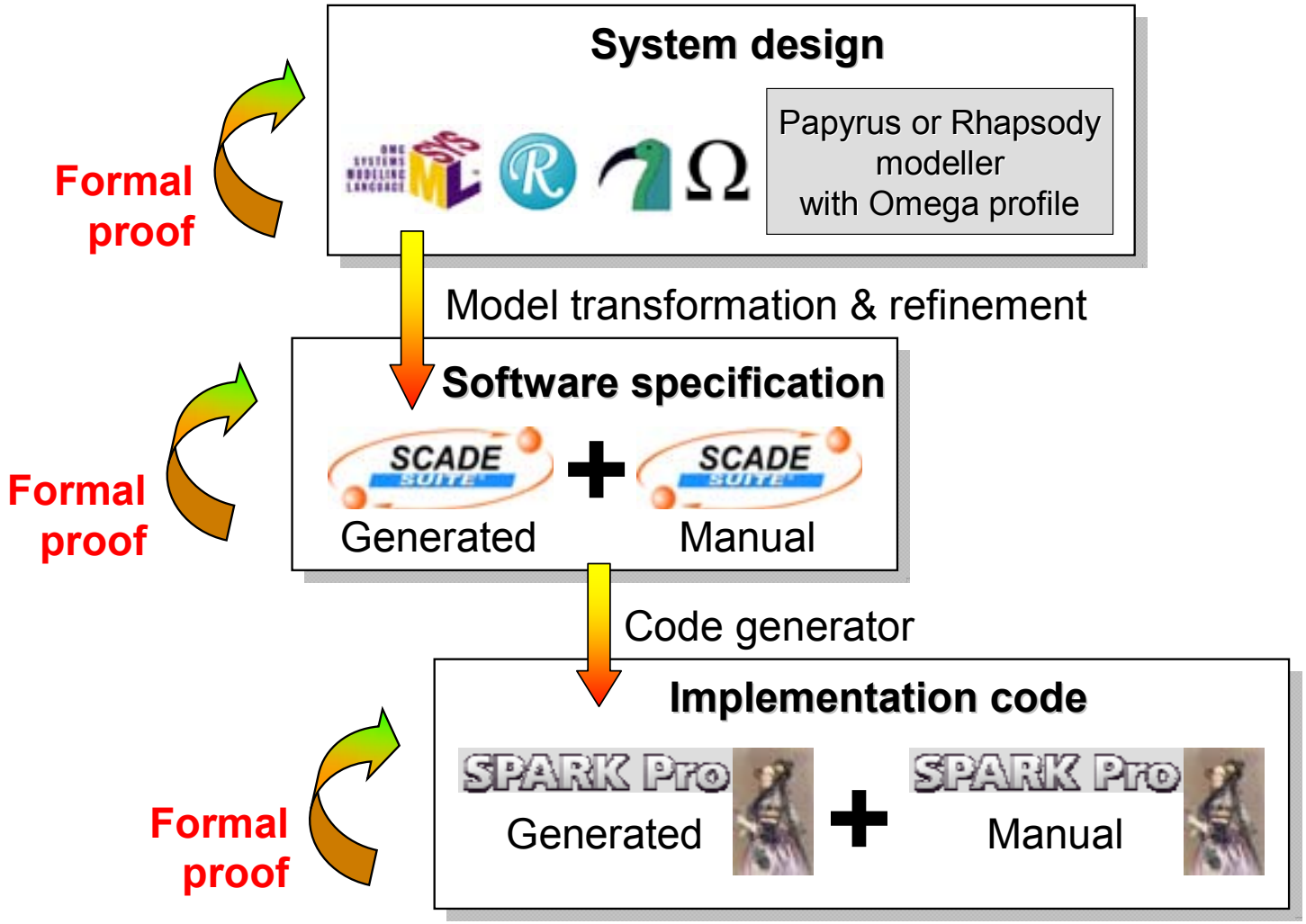
Outline

1. Full Model Driven Engineering development process
2. OMEGA SysML Profile & Toolset
3. The Automated Transfer Vehicle (ATV) case study
4. Validation results
5. Conclusions

Outline

1. Full Model Driven Engineering development process

Full Model Driven Engineering Process



This project has been partially funded by the European Space Agency.

Outline

2. OMEGA SysML Profile & Toolset

The OMEGA Language

- SysML Profile for the specification and verification of real-time embedded systems
- Consists of:

A large subset of SysML

+

Model coherence constraints

+

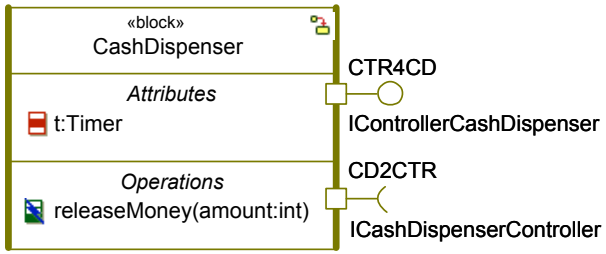
A formal operational semantics

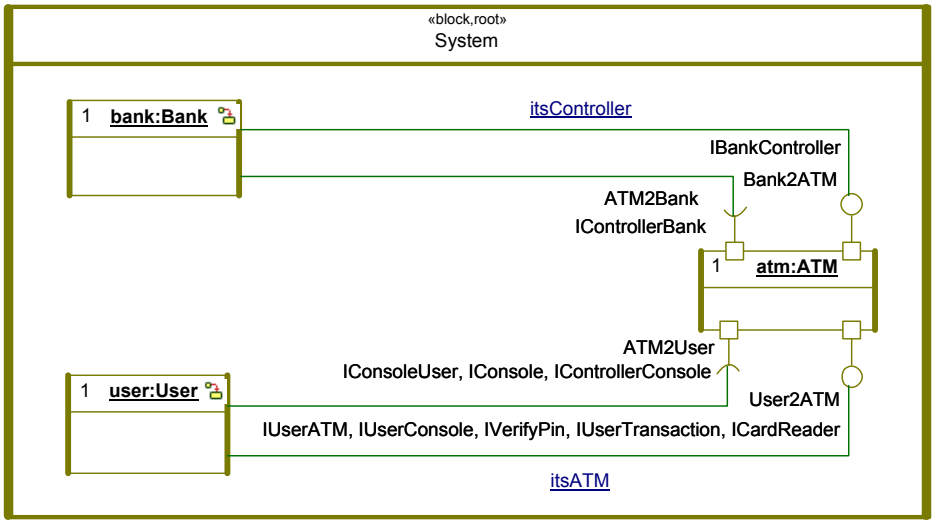
+

Real-time & verification extensions

The OMEGA Profile

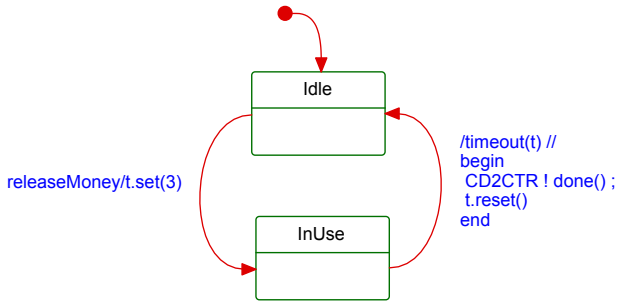
- Structure
  - SysML Block Definition Diagrams & Internal Block Diagrams
  - Blocks with properties, operations and state machines, interconnection elements and relationships
  - Structured data types and signals





The OMEGA Profile

- Structure
  - SysML Block Definition Diagrams & Internal Block Diagrams
  - Blocks with properties, operations and state machines, interconnection elements and relationships
  - Structured data types and signals
- Discrete behaviour
  - State machines
  - Asynchronous communication through operations and signals



The OMEGA Profile

- Structure
  - SysML Block Definition Diagrams & Internal Block Diagrams
  - Blocks with properties, operations and state machines, interconnection elements and relationships
  - Structured data types and signals
- Discrete behaviour
  - State machines
  - Asynchronous communication through operations and signals
- Real time
  - Clocks, time guards and transition urgency
  - Discrete or continuous specified by the user

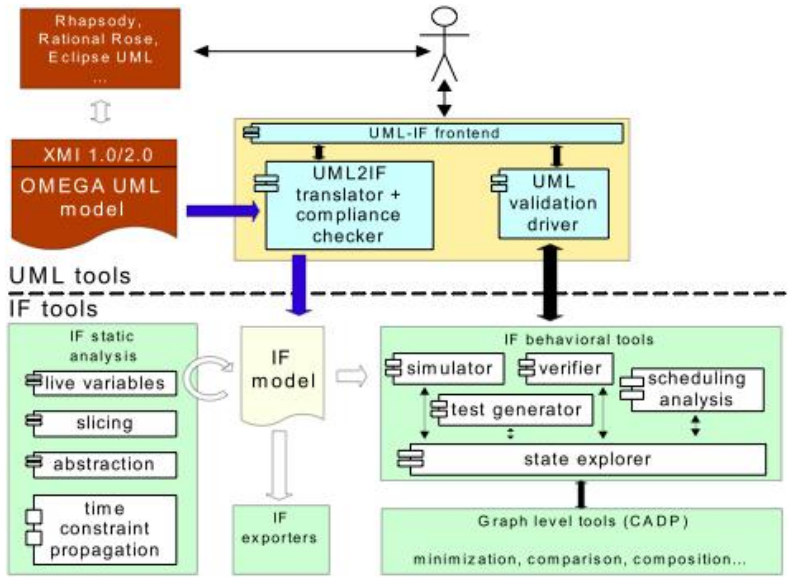
The OMEGA Profile

- Structure
  - SysML Block Definition Diagrams & Internal Block Diagrams
  - Blocks with properties, operations and state machines, interconnection elements and relationships
  - Structured data types and signals
- Discrete behaviour
  - State machines
  - Asynchronous communication through operations and signals
- Real time
  - Clocks, time guards and transition urgency
  - Discrete or continuous specified by the user
- Observers
  - Objects monitoring the system (state and events) and giving verdicts about a safety property

The IFx Toolset

- Goal: Early model validation and debugging
- Principle: Transforming to communicating extended timed automata (IF Language)

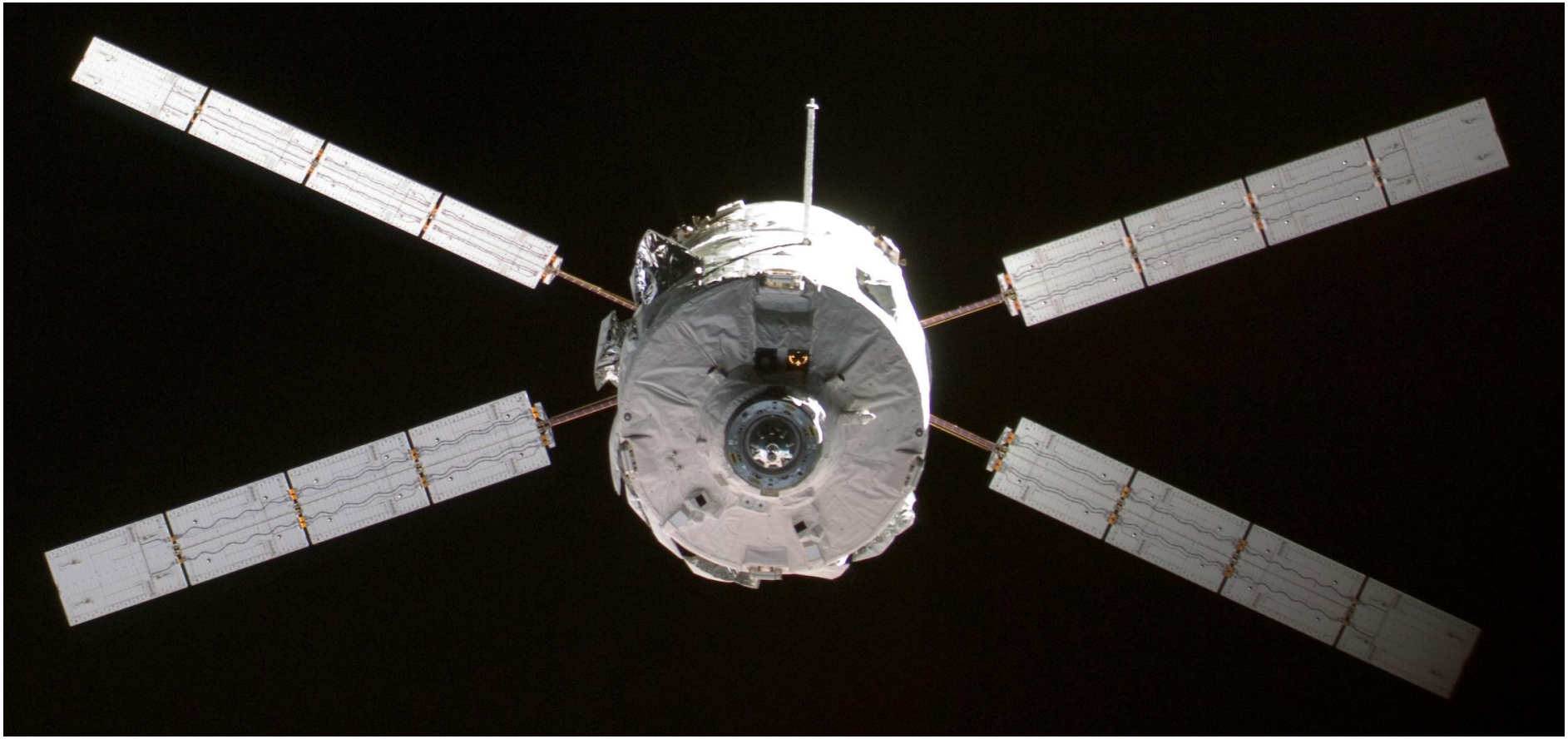
- Functionalities
  - Simulation
  - Static analysis: dead code/variable elimination, slicing, ...
  - Model-checking: observers, state graph minimization, μ-calculus, ...



Outline

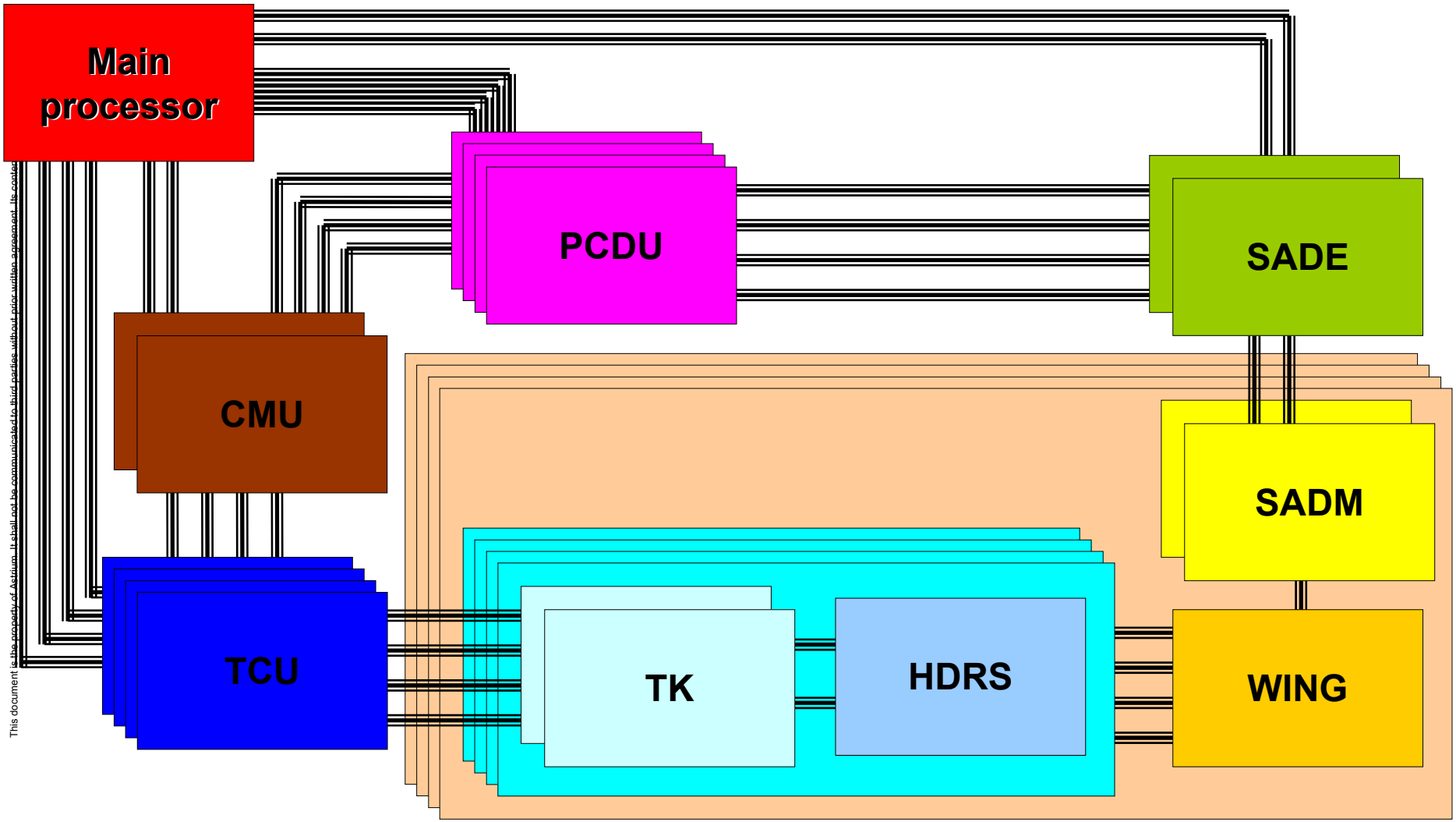
3. The Automated Transfer Vehicle (ATV) case study

The ATV Solar Generation System



The ATV has been developed by Astrium Space Transportation for ESA.

The Solar Generation System Architecture



The system model

- Reverse engineered from the actual system for the purpose of FullMDE
- 4-layer architecture
- 20 block types - HW, SW, MM - and 95 block instances
- 348 (661) ports (instances) and 372 (504) connectors (instances)

The system model

- Reverse engineered from the actual system for the purpose of FullMDE
- 4-layer architecture
- 20 block types - HW, SW, MM - and 95 block instances
- 348 (661) ports (instances) and 372 (504) connectors (instances)
- 18 interfaces for port types

The system model

- Reverse engineered from the actual system for the purpose of FullMDE
- 4-layer architecture
- 20 block types - HW, SW, MM - and 95 block instances
- 348 (661) ports (instances) and 372 (504) connectors (instances)
- 18 interfaces for port types
- 1-fault tolerant
- 62 possible hardware failures

Formal system requirement

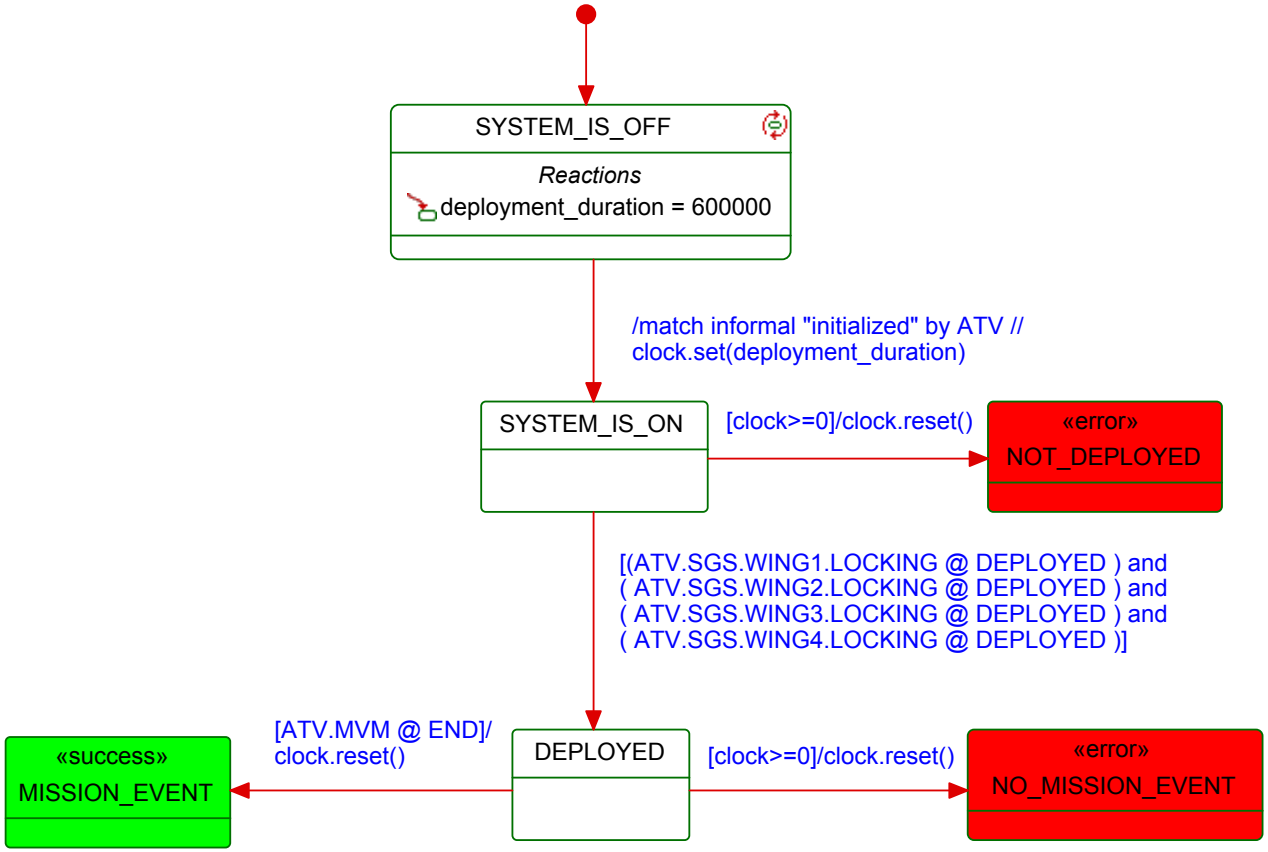
Property

After 10 minutes since SGS start-up, all 4 wings are deployed and the Mission and Vehicle Management is aware of it.

Formal system requirement

Property

After 10 minutes since SGS start-up, all 4 wings are deployed and the Mission and Vehicle Management is aware of it.



Outline

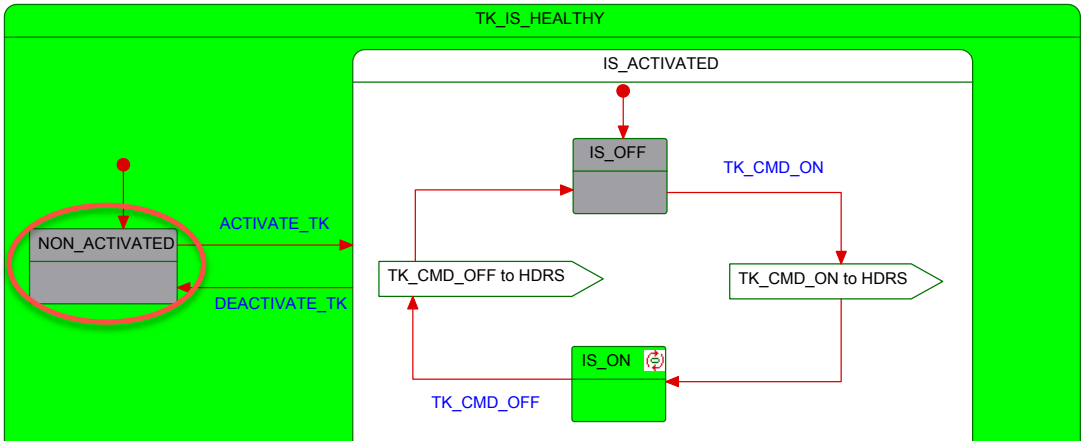
4. Validation results

Verification by simulation

- Scenario length: 2400 steps and one minute execution
- Discovered modelling errors due to reverse engineering and omitted at model review:

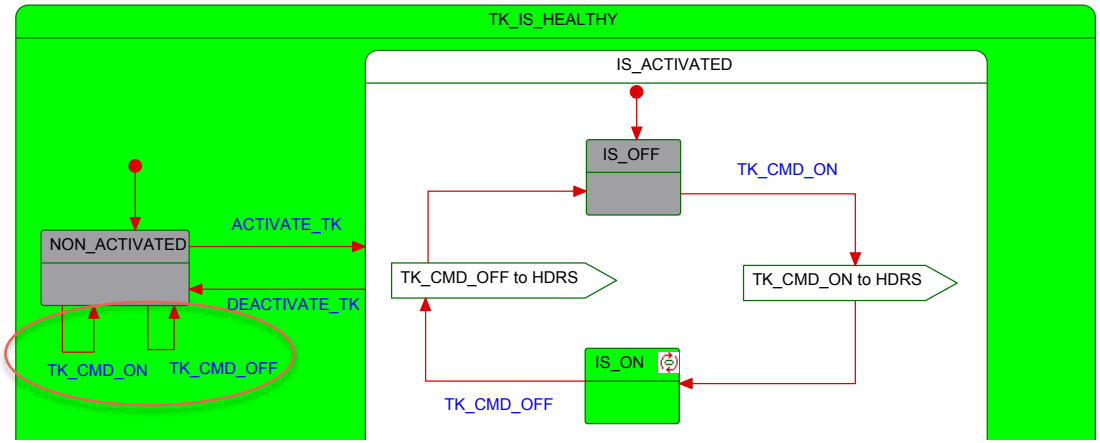
Verification by simulation

- Scenario length: 2400 steps and one minute execution
- Discovered modelling errors due to reverse engineering and omitted at model review:
  - Unexpected message receptions for wing parts



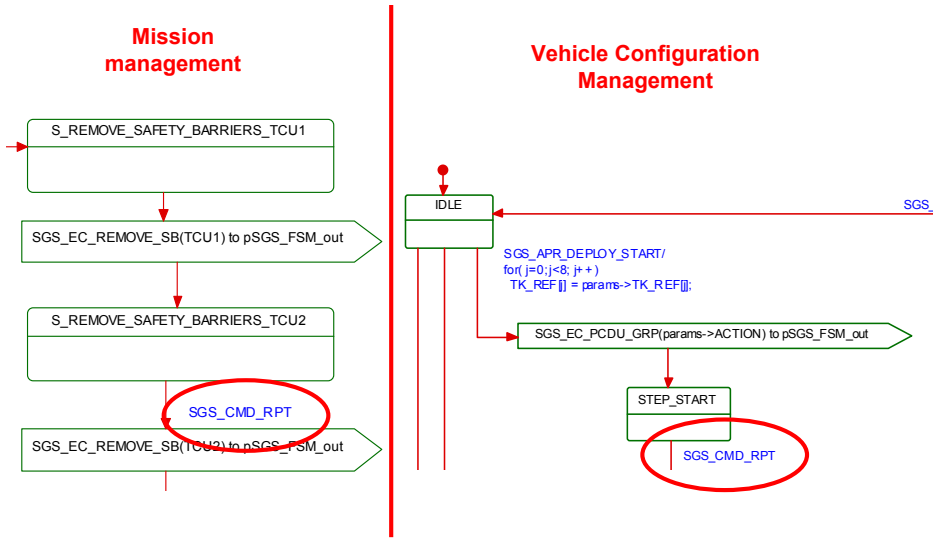
Verification by simulation

- Scenario length: 2400 steps and one minute execution
- Discovered modelling errors due to reverse engineering and omitted at model review:
  - Unexpected message receptions for wing parts



Verification by simulation

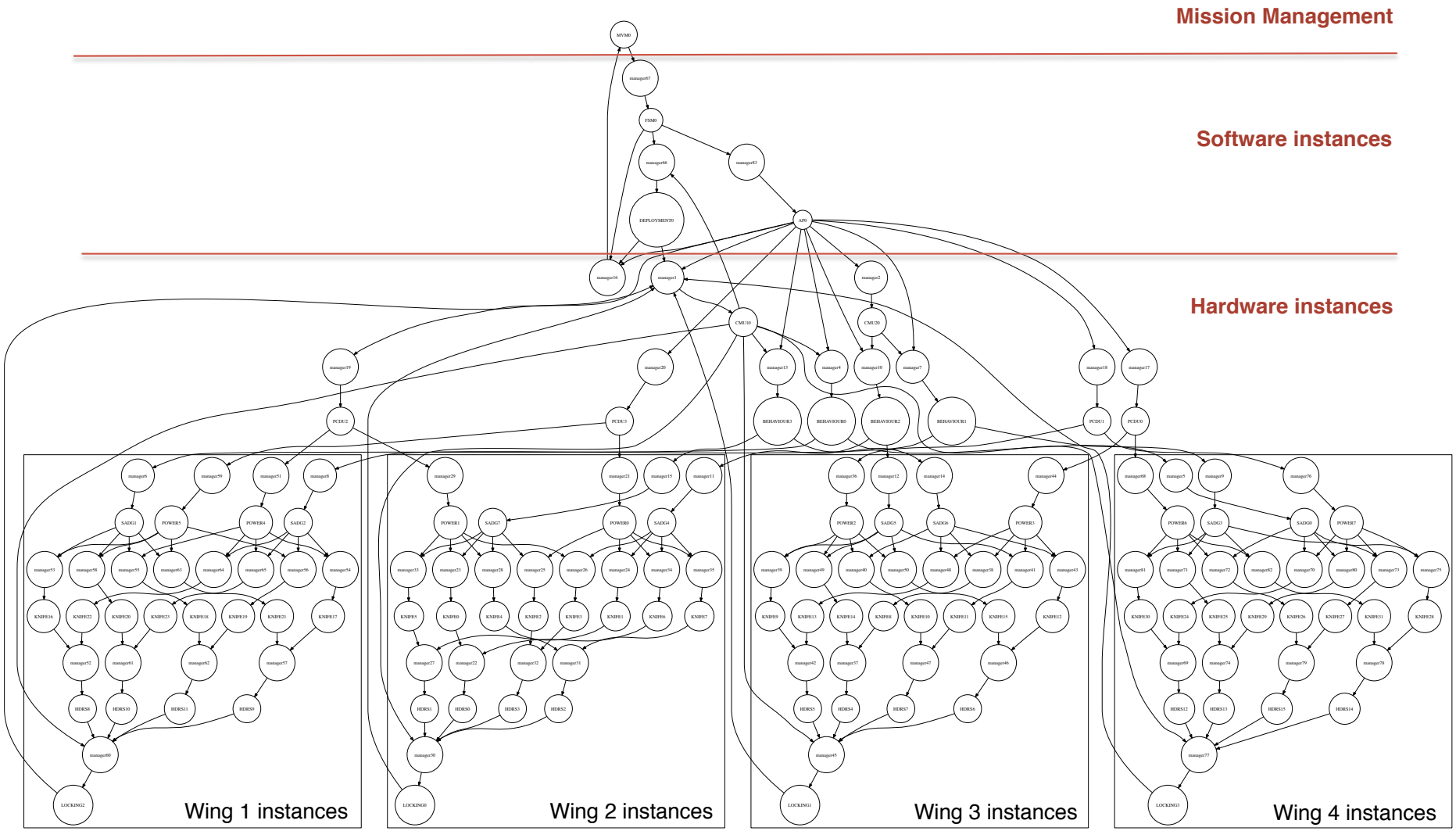
- Scenario length: 2400 steps and one minute execution
- Discovered modelling errors due to reverse engineering and omitted at model review:
  - Unexpected message receptions for wing parts
  - Ambiguous parallel receivers for Mission and Vehicle Management



Verification by simulation

- Scenario length: 2400 steps and one minute execution
- Discovered modelling errors due to reverse engineering and omitted at model review:
  - Unexpected message receptions for wing parts
  - Ambiguous parallel receivers for Mission and Vehicle Management
  - Incorrect (sequences of) requests that result in deadlocks; e.g. SADE receives deactivation before disable

State space explosion and its cause

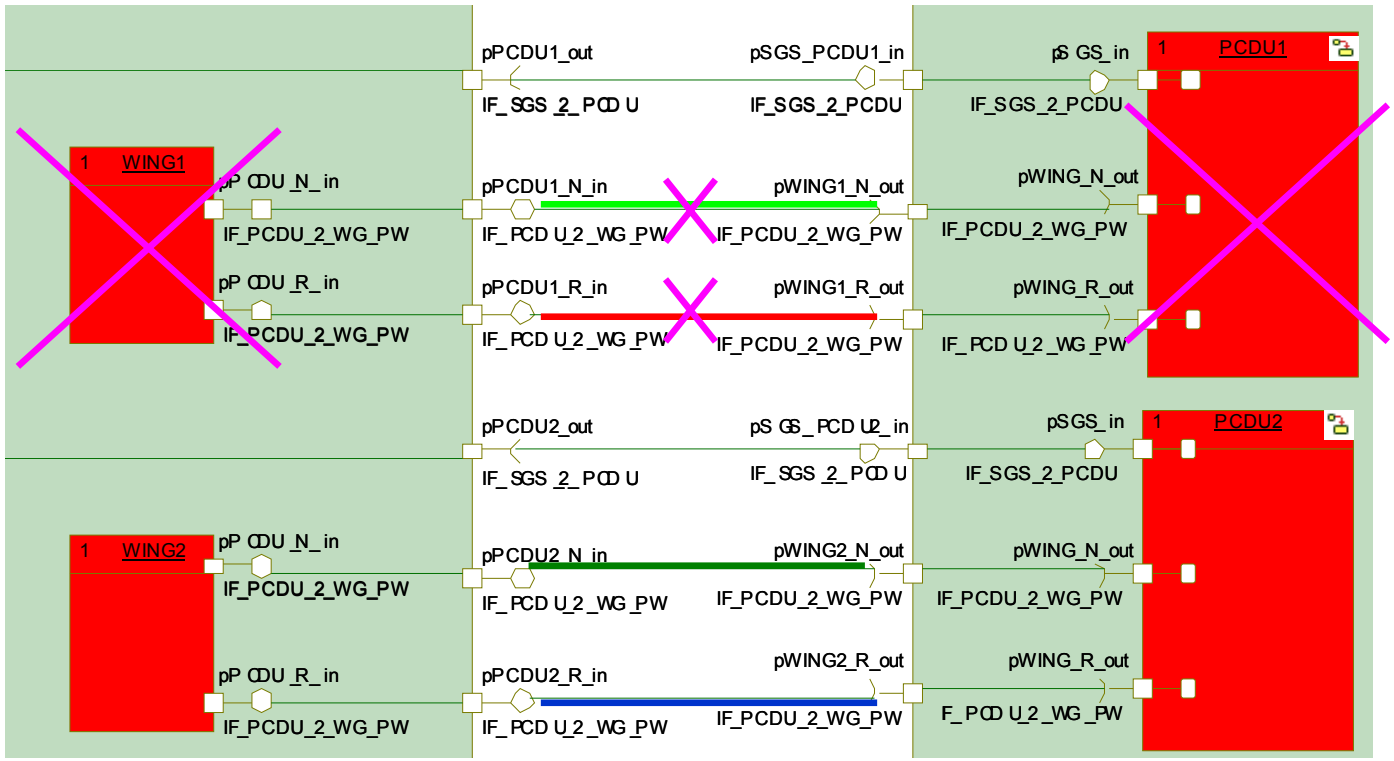


Non-exhaustive model-checking

- Executed on a single thread with a predefined scheduling for parallel actions
- Still useful for discovering logical errors:

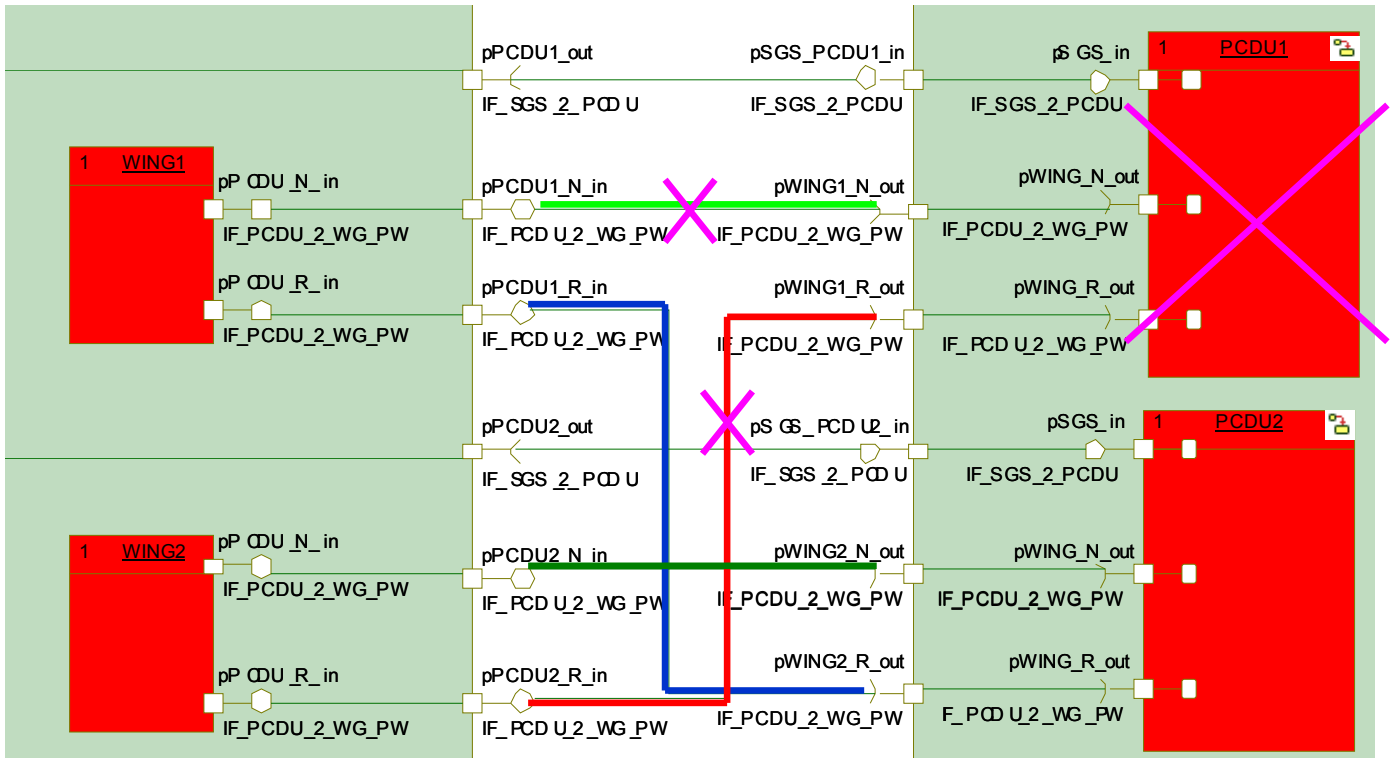
Non-exhaustive model-checking

- Executed on a single thread with a predefined scheduling for parallel actions
- Still useful for discovering logical errors:
  - Incorrect connections between the power units and the wings



Non-exhaustive model-checking

- Executed on a single thread with a predefined scheduling for parallel actions
- Still useful for discovering logical errors:
  - Incorrect connections between the power units and the wings



Non-exhaustive model-checking

- Executed on a single thread with a predefined scheduling for parallel actions
- Still useful for discovering logical errors:
  - Incorrect connections between the power units and the wings
  - Unhandled received requests by the hold-down and release systems

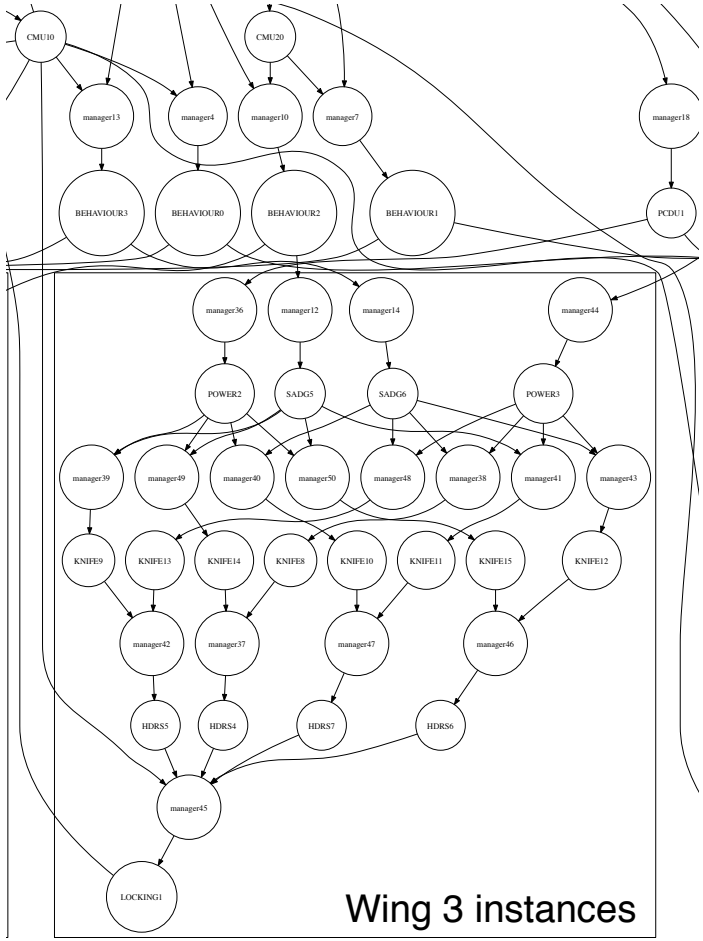
Non-exhaustive model-checking

- Executed on a single thread with a predefined scheduling for parallel actions
- Still useful for discovering logical errors:
  - Incorrect connections between the power units and the wings
  - Unhandled received requests by the hold-down and release systems
  - Control and monitoring unit is already 1-fault tolerant, which makes this type of failure incorrect and removed from the set of verifiable errors

Verification using abstractions

Abstraction

One wing structure that does not experience any hardware fault is replaced by a block with a simpler behaviour: it ends up by being deployed.



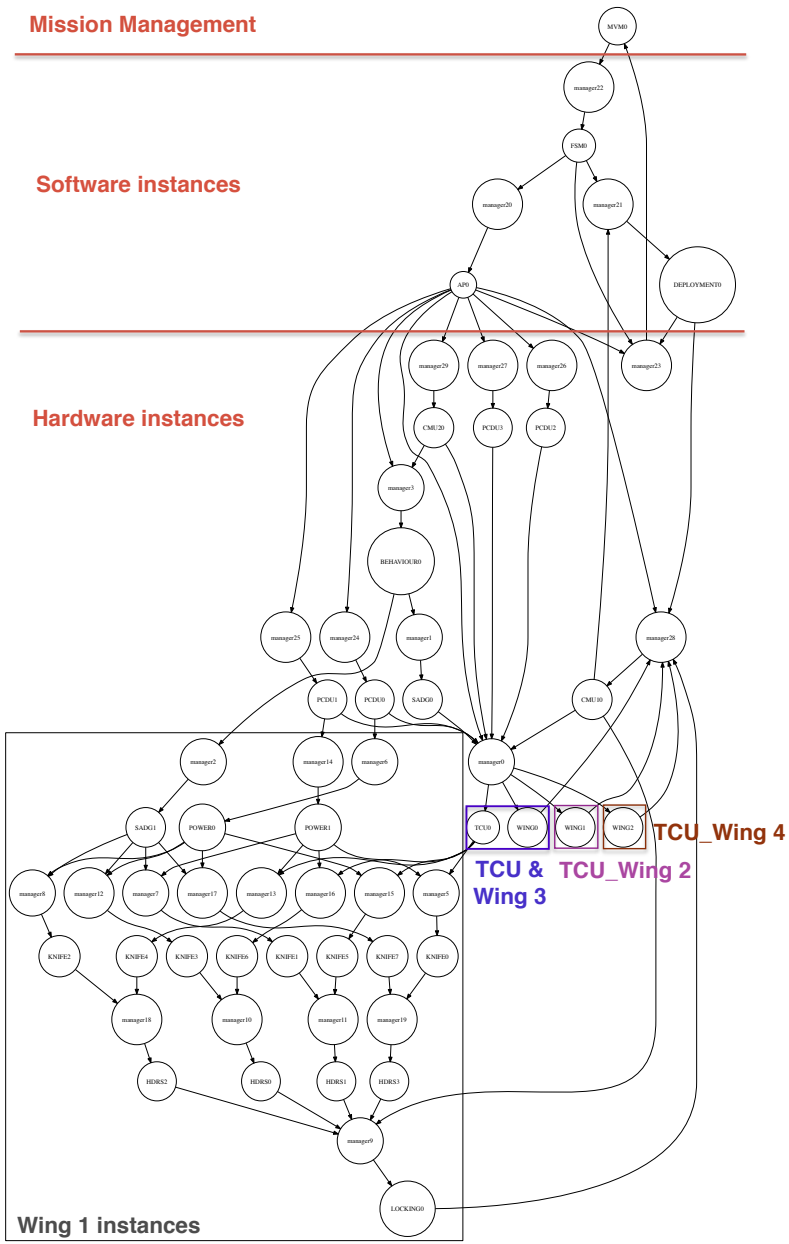
Verification using abstractions

Abstraction

One wing structure that does not experience any hardware fault is replaced by a block with a simpler behaviour: it ends up by being deployed.

- System configuration: 1 extended wing and 3 abstract ones

Abstract communication graph



Verification using abstractions

Abstraction

One wing structure that does not experience any hardware fault is replaced by a block with a simpler behaviour: it ends up by being deployed.

- System configuration: 1 extended wing and 3 abstract ones
- 4 configurations, each being manually modelled

Verification using abstractions

Abstraction

One wing structure that does not experience any hardware fault is replaced by a block with a simpler behaviour: it ends up by being deployed.

- System configuration: 1 extended wing and 3 abstract ones
- 4 configurations, each being manually modelled
- The total number of instances is reduced by 55%

Verification using abstractions

Abstraction

One wing structure that does not experience any hardware fault is replaced by a block with a simpler behaviour: it ends up by being deployed.

- System configuration: 1 extended wing and 3 abstract ones
- 4 configurations, each being manually modelled
- The total number of instances is reduced by 55%
- Separate verification for each 60 possible failures for each configuration

Verification using abstractions

Abstraction

One wing structure that does not experience any hardware fault is replaced by a block with a simpler behaviour: it ends up by being deployed.

- System configuration: 1 extended wing and 3 abstract ones
- 4 configurations, each being manually modelled
- The total number of instances is reduced by 55%
- Separate verification for each 60 possible failures for each configuration
- Error detected: failure of the redundant thermal knife while the nominal one is enabled leads to a not deployed wing

Towards Contract-Based Reasoning

- Is the used abstraction correct?

Towards Contract-Based Reasoning

- Is the used abstraction correct?
- *Assumption* about the environment of a wing wrt the order and timing of the sent requests

Towards Contract-Based Reasoning

- Is the used abstraction correct?
- *Assumption* about the environment of a wing wrt the order and timing of the sent requests
- The concrete environment has to *guarantee* this assumption given that the wings behave as described by the abstraction

Towards Contract-Based Reasoning

- Is the used abstraction correct?
- *Assumption* about the environment of a wing wrt the order and timing of the sent requests
- The concrete environment has to *guarantee* this assumption given that the wings behave as described by the abstraction

→ Both steps have been formally verified within OMEGA-IFx

Outline

Conclusions

- Modelling of a complex system design with OMEGA SysML
- Verification & Validation by simulation and model-checking
- Use of abstractions & Contract-based Reasoning

Conclusions

- Modelling of a complex system design with OMEGA SysML
- Verification & Validation by simulation and model-checking
- Use of abstractions & Contract-based Reasoning
- User feedback
  - More formal approach than the classical SysML
  - Early detections of errors in the model
  - Complexity in usage of the tool chain OMEGA-IFx
  - Proof limitations

Future Work

- Formal definition of contracts within OMEGA-IFx
- Proof automation based on circular reasoning
- Automated assumption generation