A case study in formal system engineering with SysML

Iulia Dragomir¹, Iulian Ober¹ and David Lesens²

¹IRIT - University of Toulouse

²Astrium 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
1. Full Model Driven Engineering development process
The SysML standard is not always precise enough and leaves an important part of its semantics undefined. The FMDE project has thus adapted the OMEGA profile (previously available for UML) to the SysML language in order to suppress any ambiguity in the system models and to allow formal proof. This requires the addition of new modelling constructs and the definition of a set of rules to clarify some semantics variation points of SysML (e.g. forbidding of bidirectional ports, typing of connectors, port behaviours). The behaviour of the system can be modelled in OMEGA SysML by state machines and operations invoking actions. The OMEGA profile also allows the description of timed behaviours and observers to formalise the requirements and dynamic properties of the system.

The OMEGA SysML profile is supported by a toolbox called IFx-OMEGA. IFx-OMEGA provides some features for the simulation and verification of OMEGA SysML models, relying on a model transformation from OMEGA models to the IF language.

Finally, even though Ada is well adapted for programming critical software, it still contains some ambiguous or dangerous constructs. The FMDE project has thus proposed to use the SPARK language. The SPARK language is specifically designed to support the development of software for systems where the correct operation of the software is essential for the safety or security of the system. The language is based on a core subset of Ada and augmented with annotations which describe and support the programming-by-contract approach. The SPARK Toolset offers static verification that is unrivalled in terms of its soundness, low false-alarm rate, depth and efficiency. The tools also generate evidence for correctness that can be used to build an assurance case to satisfy the requirements of industry regulators and certification bodies. The language and tools have been successfully used in the development of many high-integrity systems in various domains: air traffic control, on-board aircraft systems, control systems, rail systems, as well as security applications.

Optimisation of the successive refinement steps (from the system design to the software design and then from the software design to the implementation code) thanks to model transformation techniques:

- Automatic generation of a skeleton of the software model in SCADE Suite from the system model in SysML. This objective has been achieved by integrating SCADE Suite and the Papyrus SysML modeller through the new SCADE System Designer tool. Having two interconnected meta-models allows consistency to be kept between the system model and the software model. A model transformation creates a SCADE node with the same interface as a system component (modelled by a SysML block).
- Automatic generation of SPARK code from the software model in SCADE Suite by a tool certifiable to DO178B level A. The combination of SPARK and SCADE allows the verification of the correct integration between manual and automatically generated code, the verification of absence of runtime errors and the verification of functional properties. In order to get maximum benefit from this SPARK/SCADE combination, the SCADE language has been enhanced with new basic types (int8, int16, int32, int64, uint8, uint16, uint32, uint64, float32, float64), better management of imported types and definition of ranges (minimal and maximal bounds of a type).

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

---

```
stm [block] CashDispenser [StatechartOfCashDispenser]

Idle

InUse

releaseMoney/t.set(3)

/timeout(t) //

begin
  CD2CTR ! done() ;
  t.reset()
end
```

---

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

---

Iulia Dragomir (IRIT)

A case study in formal system engineering with SysML
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
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, $\mu$-calculus, ...
The Automated Transfer Vehicle (ATV) case study
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.
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

```
TK_IS_FAILED
TK_CMD_ON to HDRS
TK_CMD_OFF to HDRS
ERROR
Thermal knife commanded when desactivated

TK_IS_HEALTHY
NON_ACTIVATED
IS_ACTIVATED
IS_OFF
ACTIVATE_TK
DEACTIVATE_TK
TK_CMD_OFF to HDRS
TK_CMD_ON to HDRS
TK_CMD_OFF
TK_CMD_ON
FAILURE_ON
FAILURE_OFF
TK_deactivation

«Requirement»

StatechartDiagram
```
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

![Diagram showing mission management and vehicle configuration 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
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.
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%

Error detected: failure of the redundant thermal knife while the nominal one is enabled leads to a not deployed wing.
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
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
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
Conclusions
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