A Model Management Approach for Assurance Case Reuse due to System Evolution

Sahar Kokaly, Rick Salay Valentin Cassano, Tom Maibaum and Marsha Chechik

kokalys@mcmaster.ca
Airbus A400M plane crash linked to software fault

By Leo Kelion
Technology desk editor

20 May 2015 / Technology

The A400M cargo plane crashed near Seville airport on 9 May
FDA: SOFTWARE FAILURES RESPONSIBLE FOR 24% OF ALL MEDICAL DEVICE RECALLS

by Paul Roberts

Software failures were behind 24 percent of all the medical device recalls in 2011, according to data from the U.S. Food and Drug Administration, which said it is gearing up its labs to spend more time analyzing the quality and security of software-based medical instruments and equipment.
Volvo recalls 59,000 cars over software fault

20 February 2016 | Europe

Swedish carmaker Volvo is recalling 59,000 cars across 40 markets over a fault that can temporarily shut down the engine.

Sweden, Britain and Germany are the main markets affected.
The U.S. government has issued its first rules for automated vehicles. They include a 15-point set of “safety assessment” guidelines for self-driving systems. These cover issues such as cybersecurity, black-box recordings to aid crash investigations, and potential ethical conundrums on the road.
“Standards are documented agreements containing technical **specifications** or other precise criteria to be used consistently as **rules**, **guidelines**, or **definitions** of characteristics, to ensure that materials, products, processes and services are fit for their purpose.”

[ISO 1997]
DO-178B - Software Considerations in Airborne Systems and Equipment Certification.
IEC62304 – Medical device software – software life cycle processes.
Compliance

**What is it?**

The extent to which software developers have acted in accordance with practices set down in the standard.

**Why it is done?**

Establish **consistency** between actual development process and normative models embedded in the standards.

**How is it done?**

An artifact, called an Assurance Case, is often required to demonstrate that a system meets the property set forth by the standard (e.g., Safety, Privacy, Security, etc.)
Standards are great, but they are also... BIG complex
## 1. Vocabulary

### 2. Management of functional safety
- 2-5 Overall safety management
- 2-6 Safety management during item development
- 2-7 Safety management after release for production

### 3. Concept phase
- 3-5 Item definition
- 3-6 Initiation of the safety lifecycle
- 3-7 Hazard analysis and risk assessment
- 3-8 Functional safety concept

### 4. Product development: system level
- 4-5 Initiation of product development at the system level
- 4-6 Specification of the technical safety requirements
- 4-7 System design
- 4-8 Item integration and testing
- 4-9 Safety validation
- 4-10 Functional safety assessment
- 4-11 Release for production

### 5. Product development: hardware level
- 5-5 Initiation of product development at the hardware level
- 5-6 Specification of hardware safety requirements
- 5-7 Hardware design
- 5-8 Hardware architectural metrics
- 5-9 Evaluation of violation of the safety goal due to random HW failures
- 5-10 Hardware integration and testing

### 6. Product development: software level
- 6-5 Initiation of product development at the software level
- 6-6 Specification of software safety requirements
- 6-7 Software architectural design
- 6-8 Software unit design and implementation
- 6-9 Software unit testing
- 6-10 Software integration and testing
- 6-11 Verification of software safety requirements

### 8. Supporting processes
- 8-5 Interfaces within distributed developments
- 8-6 Specification and management of safety requirements
- 8-7 Configuration management
- 8-8 Change management
- 8-9 Verification
- 8-10 Documentation
- 8-11 Qualification of software tools
- 8-12 Qualification of software components
- 8-13 Qualification of hardware components
- 8-14 Proven in use argument

### 9. ASIL-oriented and safety-oriented analyses
- 9-5 Requirements decomposition with respect to ASIL tailoring
- 9-6 Criteria for coexistence of elements
- 9-7 Analysis of dependent failures
- 9-8 Safety analyses

### 10. Guideline on ISO 26262 (informative)
...this makes compliance costly

What is needed?
Ways to (semi-)automate compliance assessment activities to reduce their cost.
In a previous position paper [MiSE@ICSE’16]

- Identified compliance management scenarios:
  - Compliance with multiple standards.
  - Standard/system slicing for partial compliance checking.
  - Lifting compliance assessment from products to product lines.
  - Identifying relationships between standards.
  - Assurance case reuse

- Showed how model management techniques could be adapted and used to address these scenarios.
In this paper: Assuranse Case Reuse due to System Evolution
Contributions

1. Defined a **generic model management framework** for assurance case reuse due to system evolution.

2. Identified and specified **model management operators** required for a management approach and presented an **algorithm** for reuse.

3. Evaluated the generic framework by **instantiating** for ISO 26262 vehicle safety cases with the **KAOS** goal modeling language used for expressing assurance cases.

4. Applied this instantiation to a **power sliding door system** from the automotive domain.
Outline

- Introduction and Motivation
- Summary of Contributions
- Getting started:
  - Assurance Case Modeling
  - Model Management
- Assurance Case Impact Assessment Algorithm
  - Demonstration: Power Sliding Door
- Analysis
- Conclusion and Future Work
What is an Assurance Case?

• An artifact that shows how important **claims** about the system (e.g., requirement satisfaction) can be **argued** for, ultimately from **evidence** obtained about the system such as model checking, test results, expert opinion, etc.
Example: Power Sliding Door assurance case

SG1: Avoid activating the actuator while the vehicle speed is greater than 15 km/h

FSR1: The VS ECU sends the accurate vehicle speed information to the AC ECU

FSR2: The AC ECU does not power the actuator if the vehicle speed is greater than 15 km/h

FSR3: The VS ECU sends accurate vehicle speed information to the Redundant Switch.

FSR4: The Redundant Switch is in an open state if the vehicle speed is greater than 15 km/h.

FSR5: The actuator is activated only when powered by the AC ECU and the Redundant Switch is closed

E1: VS Sensor Accuracy Test Results
E2: Model Checking System Models
E3: Model Checking System Models
E4: Model Checking System Models
E5: Model Checking System Models
Assurance Case Modeling

• Some approaches for modeling assurances cases:
  – GSN, CAE, KAOS-based, OMG SACM…

In the paper:
  • Dependency Relations
  • Semantic Assumptions

Generic Assurance Case Metamodel
The Toolbox: Model Management (MM)

- High-level view in which entire models and their relationships can be manipulated using operators to achieve useful outcomes.

- Megamodel: a special type of model in which the elements represent models and the links between the elements represent relationships between the models.
Some Model Management Operators

- slice
- merge
- diff
- match
- bidirectional MT
- lift

Megamodel Operators (Map, Filter, Reduce) [MODELS’15]
Outline

- Introduction and Motivation
- Summary of Contributions
- Getting started:
  - Assurance Case Modeling
  - Model Management
- Assurance Case Impact Assessment Algorithm
  - Demonstration: Power Sliding Door
- Analysis
- Conclusion and Future Work
Recall: Assurance Case reuse due to system evolution

- **Challenge:** carefully managing the assurance case elements (claims, arguments, evidence) and dependencies between them.

- **Goal:** Reuse as many of the original assurance case components as possible.
Assurance Case Impact Assessment

“Heterogenous Megamodel Slicing for Model Evolution”. Rick Salay, Sahar Kokaly, Marsha Chechik, Tom Maibaum. Models and Evolution Workshop @MODELS’16

Assurance Case specific operators
Model Management AC Reuse

Impact Assessment Algorithm

**Params:** <Slice_T ; Merge_T>

**Input:** initial spec S : T, assurance case A : AC, traceability map R, changed spec S' : T, delta D = <C0a;C0d;C0c>

**Output:** Impact set estimate A_RMM, impact kind annotation k_RMM

1: R'_A ← Restrict(R, D)
2: dc ← Slice_T (S, Merge_T (d,c))
3: ac ← Slice_T (S', Merge_T (a,c))
4: C2_recheck ← Merge_AC (Trace(R, dc), Trace(R'_A , ac))
5: C2_revise ← Trace(R, d)
6: C3_revise ← Slice_AC (M, C2_revise)
7: C3_recheck ← Slice_AC (M, C2_recheck)
8: A_RMM ← Merge_AC (C3_revise, C3_recheck)
9: k_RMM (C3_recheck) ← ‘recheck’
10: k_RMM(C3_revise) ← ‘revise’
11: return A_RMM, k_RMM

**Takeaways:**
- Model Management (MM) Operators
- Adapted MM Operators for Assurance Cases
- MM Workflow
- Challenge working with ACs:
  - Dependencies
  - Argument Structure
Example: Power Sliding Door System

System S

Δ: removal of redundant switch

Evolved System S′
Power Sliding Door Megamodel

Driver Switch
- requestDoorOpen()
- requestDoorClose()

VS ECU
- getSpeed(sensed_speed)
- sensed_speed: Real

Redundant Switch
- requestSpeed()
- closed: Boolean
- sensed_speed: Real

AC ECU
- requestSpeed()
- sensed_speed: Real

Actuator
- openDoor()
- closeDoor()
- powered: Boolean
- activated: Boolean

Door
- open: Boolean

communicationsWith
- communicatesWith

p: VS ECU
- : AC ECU
- a: Actuator
- :Driver Switch
- s: Redundant_Switch

requestDoorOpen()
- sensed_speed
- [if sensed_speed <= 15 and a.powered and s.closed] a.activated = True, a.openDoor()

requestDoorClose()
- sensed_speed
- [if sensed_speed <= 15 and a.powered and s.closed] a.activated = True, a.closeDoor()
SG1: Avoid activating the actuator while the vehicle speed is greater than 15 km/h

\[ G(\text{not}(a.\text{activated} \land (\text{sensed}\_\text{speed} > 15))) \land\]
\[ a.\text{sensed}\_\text{speed}=\text{vehicle}\_\text{speed} \land s.\text{sensed}\_\text{speed}=\text{vehicle}\_\text{speed}) \]

FSR1: The VS ECU sends the accurate vehicle speed information to the AC ECU

\[ G(a.\text{sensed}\_\text{speed}=\text{vehicle}\_\text{speed}) \]

FSR2: The AC ECU does not power the actuator if the vehicle speed is greater than 15 km/h

\[ G(a.\text{powered} \Rightarrow a.\text{sensed}\_\text{speed} \leq 15) \]

FSR3: The VS ECU sends accurate vehicle speed information to the Redundant Switch.

\[ G(s.\text{sensed}\_\text{speed}=\text{vehicle}\_\text{speed}) \]

FSR4: The Redundant Switch is in an open state if the vehicle speed is greater than 15 km/h.

\[ G(s.\text{sensed}\_\text{speed} > 15 \Rightarrow \sim s.\text{closed}) \]

FSR5: The actuator is activated only when powered by the AC ECU and the Redundant Switch is closed

\[ G(a.\text{activated} \Rightarrow (a.\text{powered} \land s.\text{closed})) \]

E1: VS Sensor Accuracy Test Results
E2: Model Checking System Models
E3: Model Checking System Models
E4: Model Checking System Models
E5: Model Checking System Models
SG1: Avoid activating the actuator while the vehicle speed is greater than 15 km/h

\[ G(\text{not}(a.\text{activated} \text{ and } (\text{sensed}\_\text{speed} > 15)) \text{ and } a.\text{sensed}\_\text{speed} = \text{vehicle}\_\text{speed} \text{ and } s.\text{sensed}\_\text{speed} = \text{vehicle}\_\text{speed})] \]

FSR1: The VS ECU sends the accurate vehicle speed information to the AC ECU

\[ G(a.\text{sensed}\_\text{speed} = \text{vehicle}\_\text{speed}) \]

FSR2: The AC ECU does not power the actuator if the vehicle speed is greater than 15 km/h

\[ G(a.\text{powered} \rightarrow a.\text{sensed}\_\text{speed} \leq 15) \]

FSR3: The VS ECU sends accurate vehicle speed information to the Redundant Switch.

\[ G(s.\text{sensed}\_\text{speed} = \text{vehicle}\_\text{speed}) \]

FSR4: The Redundant Switch is in an open state if the vehicle speed is greater than 15 km/h.

\[ G(s.\text{sensed}\_\text{speed} > 15 \rightarrow \neg s.\text{closed}) \]

FSR5: The actuator is activated only when powered by the AC ECU and Redundant Switch is closed

\[ G(a.\text{activated} \rightarrow (a.\text{powered \text{ and } s.\text{closed}})) \]

E1: VS Sensor Accuracy Test Results

E2: Model Checking System Models

E3: Model Checking System Models

E4: Model Checking System Models

E5: Model Checking System Models

Assurance Case after impact assessment

Strategy: AND refinement

reuse - recheck - revise
Possible Evolved Assurance Case
(after post-processing by Assurance Engineer)

SG1: Avoid activating the actuator while the vehicle speed is greater than 15 km/h

\[ G(\text{not}(a.\text{activated} \land (a.\text{sensed}\_\text{speed} > 15)) \land a.\text{sensed}\_\text{speed} = \text{vehicle}\_\text{speed})) \]

FSR1: The VS ECU sends the accurate vehicle speed information to the AC ECU

\[ G(a.\text{sensed}\_\text{speed} = \text{vehicle}\_\text{speed}) \]

FSR2: The AC ECU does not power the actuator if the vehicle speed is greater than 15 km/h

\[ G(a.\text{powered} \rightarrow a.\text{sensed}\_\text{speed} \leq 15) \]

FSR3: The actuator is activated only when powered by the AC ECU

\[ G(a.\text{activated} \rightarrow a.\text{powered}) \]

E1: VS Sensor Accuracy Test Results

E2: Model Checking System Models

E3: Model Checking System Models

Strategy: AND refinement
Outline

- Introduction and Motivation
- Summary of Contributions
- Getting started:
  - Assurance Case Modeling
  - Model Management
- Assurance Case Impact Assessment Algorithm
  - Demonstration: Power Sliding Door
- Analysis
- Conclusion and Future Work
Analysis 1 - Soundness

• Soundness
  – limited to claims of evolution due to atom changes and deletions
  – added components required to be assessed by assurance engineer

• Future work: trace link discovery for added components
Analysis 2 – Relative Efficiency

• Relative Efficiency
  – an impact assessment approach is more efficient if it reports fewer “false positives”.
  – efficiency relies on the information the algorithm uses to determine impact (depth of knowledge about dependency relations in assurance case).

• Future work: ways to improve efficiency:
  • Exploit additional knowledge available when algorithm is instantiated for particular modeling languages.
Analysis 3 – Emergent Properties

• Emergent Properties
  – arise as a result of the integration of parts of a system
  – Two possible cases:
    • System properties
      – E.g.: “99.5% of the time, a collision when the vehicle is moving 80 kph will not result in a fatality of a passenger.”
    • Feature interaction
      – occurs when using features together results in unintended behaviour.
      – impacts due to feature interaction depend on the quality of the slicer(s).

• Future work:
  – make algorithm less conservative
  – experiment with various slicers
Outline

- Introduction and Motivation
- Summary of Contributions
- Getting started:
  - Assurance Case Modeling
  - Model Management
- Assurance Case Impact Assessment Algorithm
  - Demonstration: Power Sliding Door
- Analysis
- Conclusion and Future Work
In Summary

• Model Evolution has been studied in the MDE community
  – Assurance Case models not considered
  – Challenge: dependency relations and argument structure

• We proposed a model management approach which uses and adapts model management operators to aid the assurance engineer in the reuse of assurance cases when systems evolve.

• We analyzed approach w.r.t. soundness, relative efficiency and handling of emergent properties.

• We demonstrated approach on Power Sliding Door example from automotive domain.
Status and Next Steps

• **Currently working on tooling:** MMINT* + Compliance (ComMMINT)
  - Incorporate assurance case metamodel
  - Implement adapted MM operators to work with assurance cases (e.g., slice, merge)
  - Implement MM workflows for assurance case reuse

• The bigger picture: Model Management for Regulatory Compliance

• Case study with industrial partner to assess effectiveness
  – Possible success factor: decrease cost of re-assessing assurance when systems evolve

https://github.com/adisandro/MMINT/
Questions?

A Model Management Approach for Assurance Case Reuse due to System Evolution

Sahar Kokaly, Rick Salay Valentin Cassano, Tom Maibaum and Marsha Chechik

kokalys@mcmaster.ca