A Viewpoint-Based Approach for Formal Safety & Security Assessment of System Architectures

Julien Brunel[1], David Chemouil[1], Laurent Rioux[2],
Mohamed Bakkali[3], and Frédérique Vallée[3]

[1] Onera/DTIM
F-31055 Toulouse
firstname.lastname@onera.fr
[2] Thales Research & Technology
F-91767 Palaiseau, France
laurent.rioux@thalesgroup.com
[3] All4Tec F-53001 Laval, France
firstname.lastname@all4Tec.net

Abstract. We propose an model-based approach to address safety and security assessment of a system architecture. We present an integrated process where system engineers design the model of the system architecture, safety and security engineers specify the propagation of failures and attacks inside each component of the architecture using their dedicated tool. They also define the failure modes that have to be merged from both disciplines. The underlying analyses are then performed using Alloy. We instantiate this approach with the system engineering tool Melody from Thales, and the risk analysis supporting tool Safety Architect from All4Tec. We illustrate this work on a system that implements a landing approach of an aircraft.

1 Introduction

Safety and security are commonly identified disciplines in system and software engineering. In critical embedded system engineering, the fact to spend a lot of effort in safety engineering is a common practice, in particular because these systems generally need to be certified. Standards specify a complete and precise safety process to follow in order to be certified. More recently, architects have begun considering *security* with more attention. Indeed malicious attacks on the system may cause failures and catastrophic events. So, there is a need to not only assess the safety properties but also the security properties of critical embedded systems to create a dependable system architecture. However, the literature shows the difficulties to combine these disciplines in engineering [4].

In [3], we showed that it is possible to assess some properties of a critical embedded system architecture by using the lightweight formal language Alloy. This is a promising solution but industrial companies may not accept to require safety and security engineers to create and maintain Alloy formal models. One problem is then to rely on Alloy for formal analysis while hiding it to end-users. On the other hand, safety standards are also evolving to encourage the use of design models (MBSE, Model based

System Engineering), as well as formal techniques and tools to assess properties inside these models (MBSA, Model based System Assessment).

This article introduces a proposal for a viewpoint-based approach to integrate formal assessment with Alloy in a modelling context. Remark that we focus on the feasibility of the whole approach rather than on viewpoint-based engineering *per se* (*e.g.* overall consistency, name management, abstraction layers...) hence our approach is quite simple w.r.t. the current state-of-the-art on viewpoints [5].

Now since safety engineering and security engineering rely on specific tools, we propose a solution with 3 viewpoints:

- A design layer where system architects design the system architecture (here with the Thales in-house tool called "Melody");
- A safety/security layer where safety engineers and security engineers (based on extensions for security of the "Safety Architect" tool) can model their safety and security properties (dysfunctional and security attack model);
- A third layer which consists in a formal model (here an Alloy model) to assess safety and security properties of the system architecture.

Our approach is currently done by hand as this work is a feasability study, the purpose of which is mainly to focus on viewpoints and formal validation, rather than implementing model transformations which would be rather simple here. Notice that, as of today, the feedback of the assessment results to the system architecture design are still under study and then not addressed in this article.

2 LPV case study

This case-study is concerned by the architecting of a new Thales Avionics aircraft embedded system designed to support an LPV *landing approach*. Localizer Performance with Vertical guidance (LPV) is the highest precision GNSS aviation instrument approach procedure currently available without specialized aircrew training requirements. LPV is designed to provide 16 meter horizontal accuracy and 20 meter vertical accuracy 95 percent of the time. Its architecture is represented by Fig. 2.

We can summarize the behavior of this sub-system as follows. Two Global Navigation Satellite Systems (GPS and GALILEO) send a signal to `SBAS` processing functions. After correlations of both positions information, the `SBAS` sends the aircraft position (lateral and vertical) to two occurrences of the `LPV processing` function. The data produced by `LPV processing` functions are sent to three displays (three occurrences of a function `Acquire`). In each display, a comparison of the data received from `LPV1` and `LPV2` is performed. In case of inconsistency, an alarm is triggered by a function `Monitor`. The crew chooses which of the two LPV processings is used by each display (function `SelectSource`, not represented in Fig 2). Besides, each display receives the data computed by the other two displays. Then, the function `Crosscheck` compares the data of the current display with the data of the two others and resets the current display in case it differs from the other two displays.

This initial architecture was designed taking into account a number of safety requirements; two of these are recalled below.

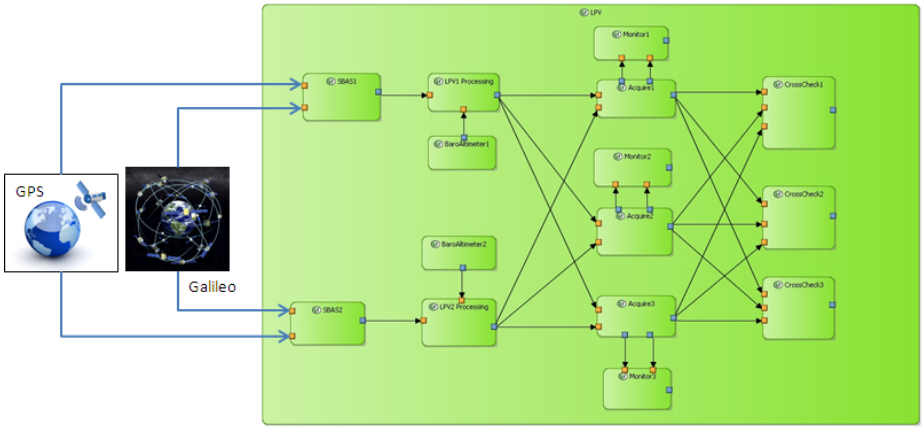

Fig. 1. LPV architecture

Safety 1 *Loss of LPV capability*. No single failure must lead to the loss of LPV capability.

Safety *Misleading information integrity*. The architecture must control the value of the LPV data provided by each calculator and between each screen and find mitigation in case of erroneous data.

We also want to ensure that the above architecture is resilient to a number of malevolent attacks. Any combination of the following attacks has been considered.

Attack 1 One malicious GPS signal (a fake signal that SBAS considers to come from GPS).

Attack 2 One constellation satellite signal is scramble.

Attack 3 The RNAV ground station is neutralized, meaning that no more RNAV signal can be send to the plane.

We will see in Sect. 5 that these requirements are easily expressible (and checked) in Alloy.

3 Model Based Safety & Security Assessment

3.1 Model Based System Engineering (MBSE)

System Engineering of aerospace electronic devices and systems (e.g. avionics, flight or aircraft systems control, mission computers . . .) is submitted to high constraints regarding safety, security, performance, environment, human factors and more; all of these deeply influence systems architecture design and development, and are to be reconciled in a relevant system architecture. The model-based system engineering (MBSE) is an efficient approach to specifying, designing, simulating and validating complex systems. This approach allows errors to be detected as soon as possible in the design process, and thus reduces the overall cost of the product. Uniformity in a system engineering project,

which is by definition multidisciplinary, is achieved by expressing the models in a common modeling language.

Due to its position of large mission-critical systems supplier for aerospace, defense & security markets, THALES invests a lot in system engineering.In particular, THALES has developed its own MBSE method named ARCADIA [11,10]. ARCADIA is based on architecture-centric and model-driven engineering activities, supported by a tool called Melody.

3.2 Model Based Safety Assessment

Model-based safety assessment is nowadays more and more considered in order to improve the safety analysis of complex systems. It relies on the idea that safety assessment activities can follow the design process in a parallel flow using the system functional and physical architectures as a common basis. The system model, either functional or physical, is used to capture the overall architecture and the interactions between its components. This abstract view of the system may be enriched with safety information using dedicated annotations in order to describe possible dysfunctional behaviors.

Safety Architect is a tool achieving risk analysis of complex systems using functional or physical architectures. Safety Architect allows the user to automatically generate the Fault Tree through a "local analysis" (see Fig. 2). The local analysis consists in linking with logical links ("and", "or") failure modes of the outputs of each component to the failure modes identified on the component inputs. During the local analysis, the user can also describe the component internal failures effects on its outputs.

In parallel, the user can also identify safety barriers that prevent the development of a single fault up to particular failure mode that could lead to a hazardous event, participating thus to the safety objectives compliance. The user must also define which failure modes of the system outputs have to be considered as hazardous events. These events are the subject of the "global analysis" provided by the tool Safety Architect.

During the global analysis, a dysfunctional simulation of the system is executed by propagating failures along the dataflow dependencies of components and until a hazardous event is reached. The results of this propagation are formulated through Fault Trees, the roots of which are all the previously identified hazardous events.

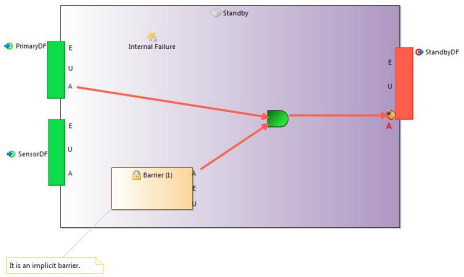

Fig. 2. Example of local analysis

3.3 Formal techniques

Formal techniques can be used to support safety and security assessment. Thanks to their mathematical foundation, they allow to prove some requirements, which provide a better confidence than more classical validation activities such as testing and manual review. A number of verification techniques have been developed over the last decades. They may differ on their expressiveness, their computational complexity and their application domain.

In this work, we have chosen Alloy [6], which is a formal system-modelling language amenable to automatic analyses. Alloy has recently been used in the context of security assessment, for instance to model JVM security constraints [8], access control policies [9], or attacks in crytogrpahic protocols [7]. Besides, we proposed in earlier work a preliminary study of the safety assessment of the LPV system with the study of a few security attacks [2,3].

The AltaRica [1] language, which is widely use for safety assessment, would have been another possible choice. However, we decided to take benefit from the model-based aspect of Alloy and its expressiveness for the specification of the properties to check. Indeed, Alloy allows to define easily the metamodel of the avionic architectures we will analyze instead of encoding them in terms of AltaRica concepts. Moreover, the specification of the properties we want to check are expressed in relational first-order logic with many features adapted to model-based reasoning.

4 Proposed approach for MBS&SA

4.1 Main principles

The approach we propose in this article consists in decoupling the system architecture model from safety & security models. This way, every engineer (be it an architect, a security or a safety engineer) can focus on her concerns solely, with dedicated tools and terminology. As of now, we chose to use two separate models: one for the safety concern and the second for the security concern. The main motivation for this separation is that safety and security domains are quite different in terms of practices, concepts used and wording. As the safety and security models rely on the system architecture model, we extract required information (e.g. functions interactions, ports and their links, data) from the architecture model and we set up initial safety and security models in Safety Architect. Starting from this, safety and security engineers complete their model by adding safety and security dysfunctional behavior. The safety and security models contain two kinds of information: the dysfunctional behavior and the properties (safety or security) to be validated. For us, a safety dysfunctional behavior represents how errors are propagated in the system architecture and a security dysfunctional behavior represent how security attacks are propagated in the system architecture. And the safety and security properties are mainly safety and security requirements that the system architecture must satisfy (e.g. integrity of the output data must be preserved even under specific attacks). Finally, these two models are combined to produce a formal Alloy model containing all the necessary input. Then, the Alloy Analyzer can formally validate the safety and security properties. If a property is violated, the Alloy Analyzer will

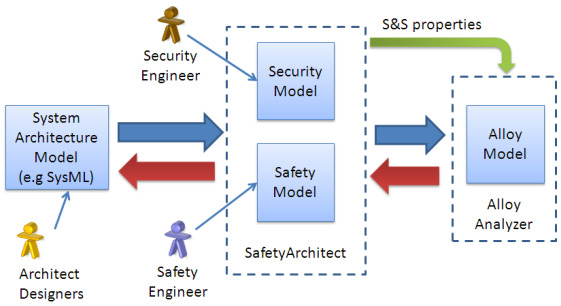

Fig. 3. Proposed approach

show a readable corresponding counter-example. This way, the engineers can identify the best way to correct the architecture to solve this identified issue.

4.2 Melody to Safety Architect

The first model transformation yields an initial Safety Architect model from the system design model. This transformation is trivial as it only reflects the structural part of the architecture. Melody functions are mapped to Safety Architect in functions. Ports give input and output ports, while data links yield data links.

4.3 Safety Architect to Alloy

We now present (a fragment of) the Alloy formalization of the language used in Safety Architect. Essentially, we define sets and relations between them: the former are called *signatures* in Alloy while the latter are described as *fields* inside the said signatures. First, we define a notion of `status` which is a signature the elements of which represent types of failures: `Absent`, `Err` (erroneous) and `Mal` (malicious) while `OK` just represents that no failure happened.

```
enum Status { OK, Err, Abs, Mal }
```

Then, blocks are mapped to functions endowed with possibly-many input and output ports as well as one status which is used to represent the notion of internal failure from Safety Architect:

```
abstract sig Function { input: set IPort, output: set OPort, status: Status }
```

Finally, ports also come with a status and can either be input or output ports. An output port may be connected to many input ports, as expressed by the field `flow`:

```
abstract sig Port { status: Status }
abstract sig IPort extends Port {}
abstract sig OPort extends Port { flow: set IPort }
```

Notice that the notion of internal failure from Safety Architect is mapped to the `status` in `Function`, although other formalizations would have been possible.

Along with these signatures, we have some Alloy *facts* which enforce static invariants on possible instances of this formalization. We do not describe them here as they are rather obvious (*e.g.* a block should have at least one port; if two ports are connected, then they should bear the same value and status...).

5 Case study evaluation

The LPV model is imported into Safety Architect modeler as shown in Fig. 4. The objective is to have either the safety view or the security view or the combination of both views as needed.

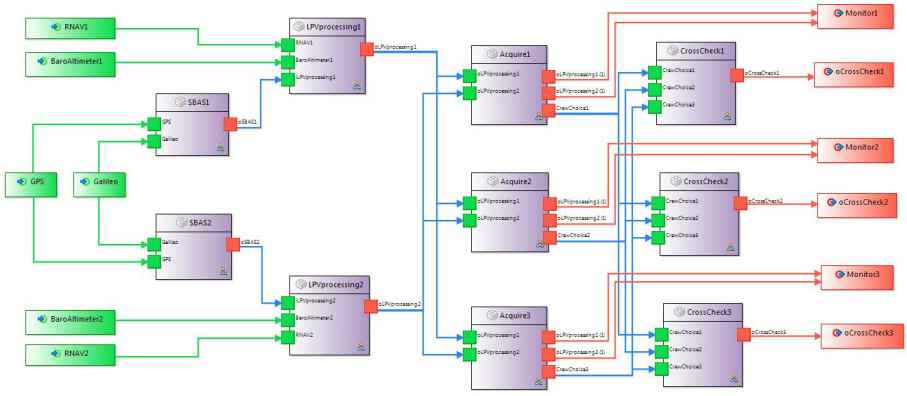

Fig. 4. LPV model in Safety Architect

5.1 Safety model

The safety analyses are based on logical equations using three generic failure modes proposed by Safety Architect on each input (in green) of a block:

A Absent (Absent data while it should be present)
E Erroneous (Non correct supplied data)
U Untimely (Data supplied while it shouldn't be)

If necessary, specific failure modes can also be defined on the input. The Untimely failure mode is defined by default but we did not use it in this case study.

For example we can say for the Safety analysis (Fig. 5, left) that one of the two ways to observe the "Erroneous" failure mode on the output "oLPVprocessing1" is to have the "Erroneous" failure mode on the input "iLPVprocessing1" *and* the "Absent" failure mode on the input "BaroAltimeter1" *or* the "Absent" failure mode on the input "RNAV1".

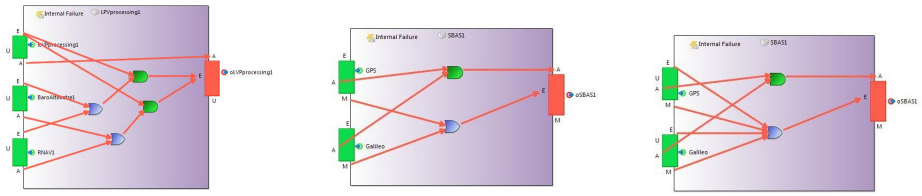

Fig. 5. Safety analysis of the block LPVprocessing1 [left] / Security analysis of the block SBAS1 (Attack 4) [middle] / Combination of the Safety and Security analysis of the block SBAS1 [right]

5.2 Security model

For security analysis, Safety Architect proposes three other generic failure modes on each input of a block:

A Absent (Absent data due to an external attack)
M Malicious (Data injected during an external attack)
E Erroneous (Malicious detected data)

Let us consider for example the security analysis of the block SBAS1 illustrated by Fig. 5 (middle). It covers "attack 4" (an attack combining attack 1 and attack 2 scenarios i.e. when SBAS considers a fake signal coming from GPS and one constellation signal is scrambled). One can see that the "Erroneous" failure mode of the output "oSBAS1" is obtained iff the "Malicious" failure mode on the input "GPS" *or* the "Malicious" failure mode on the input "Galileo" holds.

5.3 Safety and security model

The safety and security model combines both the safety and security views. By default, this is implemented as follows (but the user may modify this discretely depending on domain knowledge):

- Failure modes with the same name are identified;
- The set of logical equations of the resulting model is the union of the sets (of logical equations) of the safety and security views. However, if an equation in the safety view concerns the same output port and failure mode than an equation in the security view, there is only one resulting equation: the disjunction of both equations. The rationale is that we want to keep the two different ways for the output port to propagate the said failure.

An example is shown in Figure 5 (right). Combining the safety and security views in one model allows us to merge the two propagations into a unique propagation. The latter shows the intersection between both views; it also allows the safety or the security engineer to identify which safety or security (or both) failure modes may contribute to the appearance of a Feared Event.

5.4 Alloy code generation

We already presented in Sect. 4.3 how Safety Architect concepts (blocks, ports, failure modes) are translated into Alloy. We now show what is the Alloy representation of (an excerpt of) our case study and how to specify safety and security requirements.

Let us consider the block illustrated in Sect. 5.3 (SBAS1). Firstly, we have to declare it (as a Function) and its three ports.

```
one sig SBAS1 extends Function {}
one sig oSBAS1 extends OPort {}
one sig iGPS_SBAS1, iGalileo_SBAS1 extends IPort {}
```

We then express the connections between (ports of) functions as an Alloy constraint (a conjunction of equality between ports). Then we translate the failure propagation inside the block as Alloy facts as follows.

```
let oSBAS1 = { GPS = Abs and Galileo = Abs implies Lost
               else GMS = Mal or Galileo = Mal implies Err
               else OK }
```

Finally, we can express requirements to check directly as Alloy assertions. Note that from the identification of feared event in the Safety Architect model, we could easily generate patterns of requirements that would express that no single failure leads to this feared event, of that no attack of a certain type lead to this feared event, or that no combination of failure and attack lead to this event, etc.

For instance, the following assertion states that a fake GPS signal (attack 1 described in Sect. 2) has no bad influence on the system (the data sent by the three displays, represented by variables oSelected*i*, are still correct).

```
assert fake-GPS-has-no-bad-influence {
   (all f: Function | f.status=OK and GPS.status=Mal)
      implies oSelected1.status = OK and oSelected2.status = OK
      and oSelected3.status = OK }
```

This requirement can be verified by Alloy Analyzer with the command
`check fake-GPS-has-no-bad-influence`.

We have expressed and checked the safety requirements described in Sect. 2 and the security requirement relative to the attacks described in Sect. 2 in a similar way. It turns out that the system is robust to any single failure and to any simple attack (attack 1, 2 or 3). We also checked the consequences of any combination of two attacks: depending on the considered combination, either the system is robust or an alarm, not represented in this article, is launched. The same conclusion holds for any combination of an attack and a function failure.

6 Conclusion and future work

In this article, we proposed a model-based approach to address safety and security assessment of a system architecture. We proposed a way to make system engineers, safety

engineers and security engineers collaborate in order to perform safety and security assessment in the easiest possible way.

Now we see the feasibility and the interest of this approach, the next step is to implement it. We will need to address classical but important problems, such as the traceability between the Safety Architect models and the Melody model. For instance, after an evolution of the system architecture performed under Melody, we will have to ensure that the failure propagation inside blocks described with Safety Architect does not need to be entirely redefined.

References

1. A. Arnold, G. Point, A. Griffault, and A. Rauzy. The altarica formalism for describing concurrent systems. *Fundamenta Informaticae*, 40(2,3):109–124, Aug. 1999.
2. J. Brunel, D. Chemouil, N. Mélédo, and V. Ibanez. Formal modelling and safety analysis of an avionic functional architecture with alloy. In *Embedded Real Time Software and Systems (ERTSS 2014)*, Toulouse, France, 2014.
3. J. Brunel, L. Rioux, S. Paul, A. Faucogney, and F. Vallée. Formal safety and security assessment of an avionic architecture with alloy. In *Proceedings Third International Workshop on Engineering Safety and Security Systems (ESSS 2014)*, volume 150 of *Electronic Proceedings in Theoretical Computer Science (EPTCS)*, pages 8–19, 2014.
4. D. G. Firesmith. Engineering safety- and security-related requirements for software-intensive systems: tutorial summary. In *International Conference on Software Engineering - Volume 2 (ICSE 2010)*, pages 489–490. ACM Press, 2010.
5. IEEE Architecture Working Group. ISO/IEC/IEEE 42010 Systems and software engineering - Architecture description. The latest edition of the original IEEE Std 1471:2000, Recommended Practice for Architectural Description of Software-intensive Systems, 2011.
6. D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2006.
7. A. Lin, M. Bond, and J. Clulow. Modeling partial attacks with alloy. In B. Christianson, B. Crispo, J. Malcolm, and M. Roe, editors, *Security Protocols*, volume 5964 of *Lecture Notes in Computer Science*, pages 20–33. Springer Berlin Heidelberg, 2010.
8. M. Reynolds. Lightweight modeling of java virtual machine security constraints. In M. Frappier, U. Glässer, S. Khurshid, R. Laleau, and S. Reeves, editors, *Abstract State Machines, Alloy, B and Z*, volume 5977 of *Lecture Notes in Computer Science*, pages 146–159. Springer Berlin Heidelberg, 2010.
9. M. Toahchoodee and I. Ray. Using alloy to analyse a spatio-temporal access control model supporting delegation. *Information Security, IET*, 3(3):75–113, Sept 2009.
10. J.-L. Voirin. Method and tools to secure and support collaborative architecting of constrained systems. In *27th Congress of the International Council of the Aeronautical Science (ICAS 2010)*, 2010.
11. J.-L. Voirin and S. Bonnet. Arcadia: Model-based collaboration for system, software and hardware engineering. In *Complex Systems Design & Management, poster workshop (CSD&M 2013)*, 2013.