Landing Gear System: An ASM-Based Solution for the ABZ Case Study

Felix Kossak

Software Competence Center Hagenberg GmbH,
Softwarepark 21, 4232 Hagenberg, Austria
felix.kossak@scch.at
http://www.scch.at

Abstract. We present an ASM model for the case study given as a challenge for the ABZ'14 conference, which specifies the digital part of a landing gear system for aircraft. We strove to make the formal model well understandable for humans. We note inconsistencies, ambiguities and gaps in the case study and summarise our experiences during modelling and the proof of safety properties.

Keywords: Formal specifications, Rigorous specifications, Abstract state machines, ASMs, Safety-critical software.

1 Introduction

We herewith present experiences while working on a solution to the case study by Boniol and Wiels given as a challenge for the ABZ'14 conference [1], which specifies the digital part of a landing gear system for aircraft. Our model is based on abstract state machines (ASMs) as presented in [2]. Throughout this paper, we will refer to the cited case study document by Boniol and Wiels as the "requirements document", and by default, page numbers refer to this requirements document.

We strove to make the formal specification well understandable for humans and traceable with respect to the requirements document. We use the same terms as given in the case study document whenever possible, and in general, we use long and telling identifiers. However, expecting all stakeholders to have a technical background, we assume them to be familiar with e.g. the usual set notation.

During our work on the model and the proofs, we have noted several inconsistencies or ambiguities in the requirements document, which we state in Section 2 on *Specification Issues*.

In Section 3, we present the basic design ideas behind an ASM ground model for the software, i.e. the digital part of the landing gear system. The full model is available in [3]. Several safetey requirements according to pp. 18-19 were manually proven and one requirement was refuted; the proofs and the refutation are also available in [3].

In the final section, we state our experiences collected while creating the ASM model and proving requirements.

F. Boniol (Ed.): ABZ 2014 Case Study Track, CCIS 433, pp. 142–147, 2014.

2 Specification Issues

Formal specifications are made in order to avoid inconsistencies, ambiguities, and gaps. Thus a major part of specification work consists in the detection, documentation, and communication of such deficiencies in a given informal requirements document.

We found a few issues in the requirements document of the case study which would, in a real-life scenario, require discussions with representatives of the customer. These regard ambiguities or confusing wording as well as obvious errors. We summarise these issues in this section. A more detailed discussion can be found in [3].

2.1 Normal and Emergency Mode

We found the requirements document confusing regarding "normal" (or "nominal"?) mode and "emergency mode". And while the document states that "In this case study, we do not consider the emergency mode" (p. 1), there is an output variable "anomaly" to be set and the requirements require to set an output variable "normal_mode" to false in certain cases (see p. 19, "Failure mode requirements").

We interpret the remark on p. 3 (top), "... the green light ... must be on", such that the monitoring part of the specified system should still work even in case of "failure" (if possible). On the other hand, one can assume that the controlling part should *not* do anything anymore, i.e. should *not* send any commands to any valves anymore. In order to continue operating in principle while not sending any further commands, however, it is necessary to know whether the emergency mode is active or not, for which we use the variable "normal_mode", as mentioned. We interpret "failure" as synonymous with "anomaly".

2.2 Synchronous Parallelism

Two computing modules shall run "in parallel" (p. 5). However, it is not explicitly stated whether they shall be executed synchronously or asynchronously. We assume that they shall be executed *asynchronously*.

2.3 Obvious Errors in Monitoring Specification

We found two obvious errors in Section 4.3 of the requirements document on health monitoring, regarding *gears motion monitoring* (p. 17):

- In the first list item, it surely must read, "if the control software does not see the value $gear\_extended[x] = false$ [...] after stimulation of the retraction electro-valve [...]", instead of *gear_retracted*.
- In the third list item, it must likewise read, "if the control software does not see the value $gear\_retracted[x] = false$ [...] after stimulation of the extension electro-valve [...]", instead of *gear_extended*.

Furthermore, under *Gears motion monitoring* (p. 17), in the second, third, and fourth item, it is stated that "the doors are considered as blocked" when obviously it must read, "the *gears* are considered as blocked".

2.4 Inconsistencies in Timing

On p. 9 of the requirements document, it is stated that the analogical switch can take up to 0.8s (i.e. 800ms) to close. However, according to p. 16, an anomaly shall be detected already 160ms after the handle position has changed. We consider Section 4 of the requirements document to be authoritative (and Section 3 to be primarily informative), thus we assume 160ms.

2.5 Miscellaneous

In Section 5 (Requirements / Properties) of the requirements document (pp. 18–19), we encounter a "command button" which can be pushed "DOWN" or "UP". We assume that this is synonymous to the "handle" as mentioned e.g. on p. 2 or on p. 6.

In Section 3 of the requirements document, on p. 11, it is stated that "door cylinders are locked [...] only in closed position." This is corroborated on p. 5 (Section 2). However, on p. 6, we read that "$door\_open_i[x]$ is true if and only if the corresponding door is locked open". Likewise, in Section 5 (Requirements), e.g. in (R_{31}) (p. 18), there is talk of "when the three doors are locked open". In this paper, we consider "open" and "locked open" as synonymous in the context of doors.

3 A Ground Model for the Software

We developped an ASM ground model for the digital part of the landing gear system which is detailed in a technical report [3]. We laid an emphasis on traceability and general understandability, having experienced that lack of understandability for lay people, including domain experts, managers and potentially also lawyers, is a major deterrent for the use of formal methods in practice. Therefore we also use *long* names for rules, functions, and local variables rather than single letters or short abbreviations – except from abbreviations used consistently in the requirements document as well (such as "EV" for "electro-valve"). However, as a compromise with the need for brevity and a clear structure, we use common set and set operator notation, assuming that the major stakeholders in this case have a technical background.

The main part of the "normal mode" specification is given as enumerated lists of steps, for the "outgoing" and "retraction" sequences, respectively (pp. 14–15). To render our model fully traceable, we have decided to retain the structure given in the requirements document, including the step numbers. This led to admittedly long rules, in which the transitions between possible states of the landing gear are explicitly reflected, including transitions between the outgoing and retraction sequences. A snippet from one such rule may illustrate this:

```
rule OutgoingSequence(moduleNumber) =
  if EvaluateSensor(moduleNumber, handle) = down and
     EvaluateSensor(moduleNumber, analogical_switch) = closed then
    if state(moduleNumber) ∈ {lockedRetracted,
       retract_8_generalEValveOpening} then
    parallelblock
      CloseGeneralEV(moduleNumber)
      state(moduleNumber) := extend_1_1_generalEValveClosing
    endparallelblock
    else if state(moduleNumber) =
       extend_1_1_generalEValveClosing then
         ...
```

Apart from the given number of steps and some necessary intermediate steps – we have to distinguish between e.g. "opening" and "open" states – the number of possible transitions from an outgoing state to a retraction state contribute to the length of the rules (almost 3 pages in LNCS format per rule). Note that even the first step of the outgoing sequence can be made starting from an intermediate step of the retraction sequence. However, the original structure of the requirements is clearly visible this way.

An alternative for so long rules would have been a graphical notation for *control-state ASMs* as introduced in [2]. However, when you have a graph with 33 nodes and some 55 edges, one would need a large sheet of paper to keep this legible (including sensible state names), and even then it must be questioned whether this would yield more overview. Additionally and more generally, while a graph *may* indeed give a better overview in many cases, we think that it can be more easily misinterpreted when it comes to details, and it is much easier to accidentally overlook a part of it and forget to implement it. Therefore we decided against this option.

The monitoring part is much easier to modularise:

```
rule MonitorSystem(moduleNumber) =
parallelblock
  CheckSensors(moduleNumber)
  CheckAnalogicalSwitch(moduleNumber)
  CheckPressureSensor(moduleNumber)
  CheckDoorsMotion(moduleNumber)
  CheckGearsMotion(moduleNumber)
endparallelblock
```

A further note may be due on the modelling of temporal behaviour. According to the requirements document, only linear behaviour has to be modelled, which we do by simply subtracting points of time which are set using a monitored function "now". We think a naïve reliance on some system time to provide "now" is absolutely sufficient for the specification of the given requirements, and no further constraints are required. In such a closed system, we regard system time as a primitive available for the specification language. Constraints (axioms)

concerning the "Time" universe may be required for the use of certain tools (in particular, theorem provers), but not for manual proving as we undertook.

4 Experiences and Conclusion

Modelling. Modelling the use case as an ASM was straight-forward, basically starting with the specification of the two different control sequences (pp. 14–15 of the requirements document) and then using stepwise refinement for larger steps and checks and later to include timing constraints as well as extra action required for health monitoring. Due to the flexibility of the ASM method and language, we met no method-specific obstacles.

Modelling temporal behaviour posed no problem in the given case due to assumed linear behaviour, as already mentioned in Section 3. (Non-linear behaviour would certainly pose a considerable challenge.)

We believe that we could also demonstrate that an ASM-based specification can be made well understandable also for people who are not familiar with this method or other rigorous software specification methods, or software development in general.

The lack of a *general* (i.e. not tool-specific) editor which can do at least simple syntax checks and identifier management was felt, but it was not seriously impeding work. In one case, however, a syntax checker would have prevented an error in the model which we only detected later when we were proving requirements.

Note that we advise against writing *specifications* in the language of a tool such as CoreASM due to (a) the restrictions of the specific language and (b) tool-specific overhead which is not necessary for human understanding and actually can be a bit irritating for non-expert stakeholders. Furthermore, a specification is typically part of a contract and should therefore be available in usual and printable document form. We have argued this case in detail in [4]. (This does not invalidate tools for other purposes, however, including validation!)

Errors in the Case Study. As can be seen in the section on *Specification Issues*, we have detected several obvious errors, inconsistencies and ambiguities in the original requirements document, even without interaction with other stakeholders. This is a common experience for us and once again documents one of the many advantages of formal specifications – in this case, of *the process of preparing* a formal specification. Even though it is not unlikely that a developer would have discovered these errors as well, it is much less likely that the necessary changes would have found their way into the specification, leading to a (possibly even undocumented) inconsistency between specification and implementation. Moreover, in a real-life setting, a developer might find it much more difficult to contact a relevant person of the customer than a specifier during the specification process.

Proving. We kept to manual proving. Proving (or refuting) a normal mode requirement was straight-forward, manually parsing (i.e., simulating) the relevant

steps of the ASM. Manual proving went relatively fast, probably *much* faster than if we had used an automated theorem prover (even when we do not count the necessary translation into a respective language). The longest proof which we performed, of (R_{41}), took us about two working days (leading to more than 11 pages of output); the other proofs took considerably less time.

Proving a failure mode requirement was more complex, as several different parts of the ASM model are relevant: Amongst others, it is necessary to pick relevant lines somewhere within the very long rules *OutgoingSequence* and *RetractionSequence* as well as look at a rule in the *Monitoring* subsection and auxiliary rules in both the *Control* and *Auxiliary Subrules* subsections. Doing this manually is certainly prone to error, and will also make it hard for reviewers to check. And while it can be doubted that a tool could provide more oversight in a *printed* proof, it could certainly help to avoid errors such as missed, relevant lines in the model and make it easier to trace the proof for those who have the tool available.

It will be very interesting to compare such manual proofs with automated or tool-checked proofs of the same problems, especially with respect to time and legibility. For comparison, compiling the complete model for both control and monitoring as well as the proofs of requirements (R_{11}bis), (R_{21}), (R_{22}), (R_{31}), (R_{41}), (R_{61}), and (R_{71}) and the refutation of (R_{11}) (cf. pp.18–19 of the requirements document) took us estimatedly a bit more than one person month.

Conclusion. We have shown that the given case study can be modelled and verified with ASMs without tool support within reasonable time and effort, although tool support would have been helpful to some degree.

Acknowledgement. This publication has been written within the project "Vertical Model Integration". The project Vertical Model Integration is supported within the program "Regionale Wettbewerbsfähigkeit OÖ 2007-2013" by the European Fund for Regional Development as well as the State of Upper Austria.

References

1. Boniol, F., Wiels, V.: The Landing Gear System Case Study. In: Boniol, F. (ed.) ABZ 2014 Case Study Track. CCIS, vol. 433, pp. 1–18. Springer, Heidelberg (2014)
2. Börger, E., Stärk, R.: Abstract State Machines - A Method for High-Level System Design and Analysis. Springer, Heidelberg (2003)
3. Kossak, F.: Landing gear system: An ASM-based solution for the ABZ 2014 case study. Technical Report SCCH-TR-1401 with the complete model and proofs (2014), `http://www.scch.at/en/rse-news/landing_gear`
4. Kossak, F., Mashkoor, A., Geist, V., Illibauer, C.: Improving the understandability of formal specifications: An experience report. In: Salinesi, C., van de Weerd, I. (eds.) REFSQ 2014. LNCS, vol. 8396, pp. 184–199. Springer, Heidelberg (2014)