Validation of Regulation Documents by Automated Analysis of Formal Models*

Didier Bert[1], Fabrice Bouquet[2], Yves Ledru[1], and Sylvie Vignes[3]

[1] LSR-IMAG, Grenoble, France
[2] LIFC, Besançon, France
[3] GET/ENST, Département Informatique et Réseaux, Paris, France

Abstract. The security of civil aviation is regulated by a series of international standards and recommended practices. The EDEMOI project aims at investigating different techniques to analyse these standards. In this paper, we address two automated analysis techniques. First state-transition diagrams are extracted to visualize the nominal behavior of the involved actors. Then, models are lightly altered, and test scenarios are generated to determine how security measures could be breaked.
Keywords. Civil aviation standards, security, modelling, formal methods, test generation.

1 Introduction

The security of civil aviation is regulated by a series of international standards and recommended practices. The main reference is Annex 17 to the Convention on International Civil Aviation [1]. The EDEMOI project [6] aims to provide different techniques to analyse these standards and to develop tools to study the consistency of regulation documents. Standards are written in English and it is not easy to ensure that some parts of the document are not in conflict with other parts or, more commonly, that some cases are not forgotten in particular situations. Moreover, the documents are updated frequently, and it is necessary to verify that the security level of a version is not lowered in the next version.

However, it is difficult to automatically validate these documents. So, the approach of the EDEMOI project is to represent regulation documents by several models which are closely related. Among them, formal models are useful to make standards more precise and to analyse them by means of tools issued from software engineering methods.

In this paper, we present two complementary automated techniques applied to the formal models. First, we consider state-transition diagrams that can be extracted from the models to visualize the nominal behavior of the involved actors. Then, models are lightly altered, and test scenarios leading to an inconsistent state are generated. They help to determine how erroneous and potentially dangerous scenarios can occur. Section 2 gives an overview of the context of the

* This work was done in the EDEMOI project of program "ACI : Sécurité Informatique" supported by the French Ministry of Research and New Technologies.

project. Section 3 shows how state-transition diagrams (statecharts) are built. Section 4 indicates how scenarios are generated by using testing techniques.

2 Context of the EDEMOI project

The project is dedicated to the investigation of regulation about airport security. The airport is the place where passengers and their baggage are controlled before boarding an aircraft. Preventing dangerous objects from being brought on board an aircraft is a significant step to prevent acts of unlawful interference during a flight. The main general document (Annex 17 [1]) must be followed by all countries that are members of the International Civil Aviation Organization (ICAO).

In the litterature, regulation modelling has not attracted much attention from the computer science research community. We have developed a requirement engineering (RE) method by extending or adapting existing RE methods. Indeed, our goal is not the development and production of software: it is to exhibit the security requirements described in the standards in order to specify the whole system and to check that the specification meets these requirements by using formal proofs. A goal-oriented requirements method such as KAOS [5] is the most appropriate. The main goal is the security of the flights. It can be decomposed into several subgoals (with respect to the airport security) such as: no dangerous object is in cabin, no dangerous object is in hold. Again, to prevent dangerous objects for being in cabin, other subgoals must be satisfied: screening of passengers and cabin baggages, no contact between screened passengers and non-screened, and so on. Once these relationships on security properties are established, the process consists in modelling situations and actors of an airport, where the subgoals are brought into play: registration desk, screening checkpoint, boarding-room, passenger boarding, etc. In all these situations, the model must describe which "events" can occur, and under which conditions the security levels are respected.

We have used UML diagrams as the representational schema for our product models (requirements and system models). Although these diagrams do not convey all the details of Annex 17, they help in structuring the models and, above all, provide a support for the discussion with certification authorities and for further validation of the models. The requirement engineering process of EDEMOI has been explained and detailed in [8]. It is pictured in Fig. 1.

The second element of modelling relies upon formal models. In the considered part of the project, formal models are written in B [2], a method dedicated to software development of critical software. Formal specifications provide tools to check the consistency of models. They can also be used as a starting point for animating specifications [4], for the generation of state transition diagrams [3] and for a test generation process [9].

As it is recognized, formal methods and their associated mathematical language are difficult to understand. So, in the EDEMOI approach, graphical and

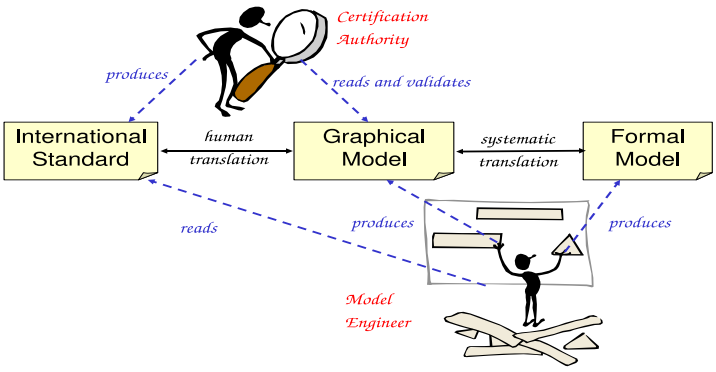

Fig. 1. Documents and actors of the modelling process.

formal models are built together, and are partially related by tool translation, as, for example, the tool presented in [7].

3 Building state-transition diagrams

A B model is a *system* defined by a component which contains *(i)* static declarations, as the global sets of baggages, passengers, and the specification of static relations and functions of the domain; *(ii)* dynamic entities, as the set of flights, passengers currently in the airport, screened passengers, relations between passengers and their flight, baggages of the passengers, etc. They constitute the *state* of the component; *(iii)* a sequence of *events*, which define the behavior of the entities. An event can be executed when its *guard* is true (a guard is a boolean condition on the state).

Security properties are modelled as *invariants* of the system. For example, the main security property is that there is no dangerous objects in an aircraft, unless these objects are given an explicit authorization. This property is mathematically stated in the model and one can *prove* that it is preserved by the events.

The airport has been modelled following Annex 17. For brevity, we do not detail this model in the paper. The events are:

- management of the flights: open_boarding, close_boarding,
- introduction of categories of passengers: new_originating_passenger, new_transit_passenger, new_transfer_passenger
- actions of the passengers in the airport: check_in_registration, passing_the_screening_point, passing_the_control_point, mixing_or_contact, boarding_in_cabin.

The "GeneSyst" tool [3] can automatically extract a state-transition diagrams from the formal model. Fig. 2 is an example of such ST diagram. It describes the set of transitions of a passenger called zz from a state where zz enters the airport (first state after initialization), then zz is considered as either

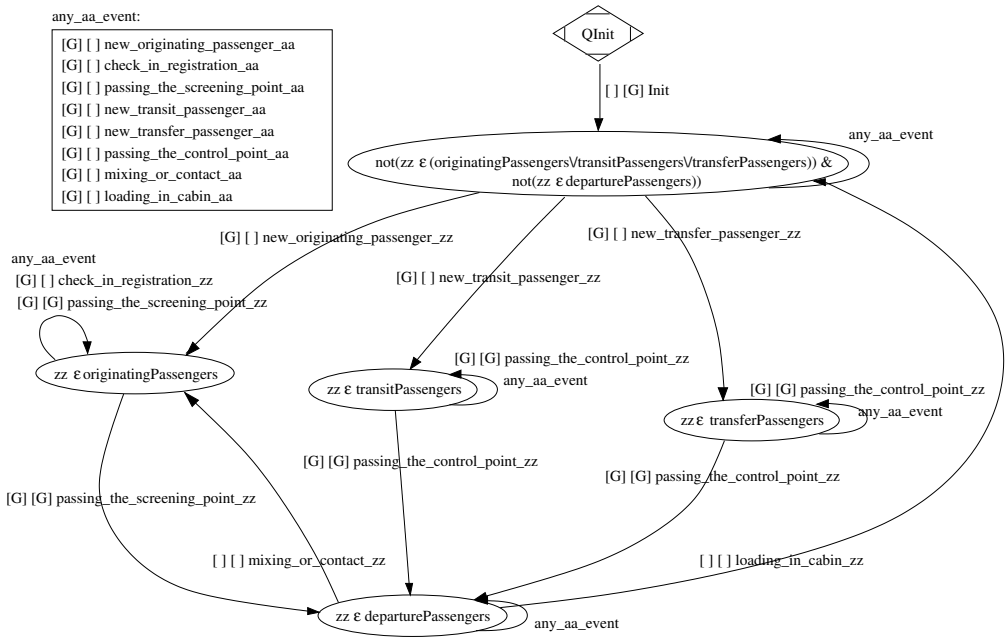

Fig. 2. Transition diagram generated by GeneSyst for passenger zz.

originating or in transit or in transfer. After that, zz is controlled or screened according to her/his characteristics, and zz becomes a departure passenger, if the checks succeed. Finally, zz is on board (not represented as a state). Each event is split into the cases where the passenger is zz (postfix of event is "_zz") or not (postfix of event is "_aa"). During the evolution of passenger zz, some events can occur for managing other passengers than zz (factorized as "any_aa_event", put for the events in the square box). For this generation, passenger zz (a new fresh variable) is introduced in the system. The resulting diagram is shown in Fig. 2.

On the diagram, a transition like $[][]event$ between $state_1$ and $state_2$, means that $event$ is always enabled in $state_1$ and that it always reaches $state_2$. A transition like $[G][]event$ means that $event$ is enabled in $state_1$ under some condition and similarly $[][G]event$ means that $state_2$ is reachable from $state_1$ by $event$ under some condition. By clicking on the transitions, the GeneSyst tool displays the involved conditions.

For the construction of ST diagrams, the states of interest are provided by the user. So, various views of the model can be explored following the entities examined in the states. An application of this generation is to check if the formal model specifies the expected behavior of the actors. If some transitions seem dubious or missing, the user can verify in the formal model and in the initial documents if this fact is due to omissions or inconsistencies, or possibly, to a misunderstanding of the regulation documents.

The GeneSyst tool can take into account refinement levels. It then produces hierarchical diagrams that represent refinement of data or actions. For instance, one can refine the previous model to distinguish between several kinds of passengers, such as ordinary, obliged and armed passengers. The specific events related to these passengers become new transitions inside old states. Another example of action refinement is the decomposition of the screening check-point into several more elementary steps.

4 Generating test scenarios

Another useful analysis technique is to determine whether, when and where a given *condition* is really mandatory to garantee the expected security level. If it is so in the formal model, then the condition must effectively be checked in the real world of the airports.

To perform this analysis, each event is transformed in such a way that each conjunct of its "guard" is replaced by its negation (to force the cases where really the condition does not hold). So, an event may produce several modified events. This transformation builds new models that contain a lot of scenarios: e.g. passengers can move without fulfilling all the conditions wrt. the initial model. From these models, the BZ-TT test generation tool [9] is able to automatically generate sequences of events starting from a given state and reaching an expected event where the state is inconsistent. A typical exemple of scenario is to search how (i.e. through which sequence of events) a given originating passenger can enter the cabin of an aircraft without satisfying the security conditions. On the set of all the generated scenarios, only those where a regular behavior can lead to a modified event and then produces an inconsistency, are selected. In that case, the condition which is falsified is really mandatory to avoid the erroneous detected scenario. The usefulness of such a generation is that it is automated and exhaustive.

For instance, in a very simple case generation process, we determined that an originating passenger can pass through the control-point instead of the screening-point. For the adequate values of the variables determined by the tool, the sequence is:

$open\_boarding$;
$new\_originating\_passenger$;
$check\_in\_registration$;
$bad\_passing\_the\_control\_point$

where the test which fails is that the passenger must be in transit or in transfer. Another erroneous sequence shows that a transfer passenger can go on board without passsing the control-point, and so on. In each case, it is easy to determine what is the point which fails in the scenario. We expect that we shall be able to detect fine-grain security conditions on more refined models.

The tests generation technique needs to be improved and adapted to our goals, but it is already promising. The determination of mandatory conditions

could be a guide to generate check-lists for the control of airports by security inspectors. Indeed, completeness of the check-lists is a concern of the ICAO experts.

5 Conclusion

The EDEMOI project is dedicated to modelling regulation documents in order to analyze the resulting models by tools issued from software engineering techniques.

In this short paper, we have presented the requirements engineering approach adapted to the kind of documents we have to deal with. We have shown two techniques developed in the project, based on the B models. The first one is the generation of state-transition diagrams by the GeneSyst tool. The second one is the generation of faulty scenarios to detect security holes. We believe that these approaches can be useful to validate models, to detect omissions or inconsistencies, to ensure non-regression in case of evolution or modification of the regulation documents, and to provide help for writing security check-lists.

References

1. A17. *Annex 17 to the Convention on International Civil Aviation - Security - Safeguarding International Civil Aviation against acts of unlawful interference.* ICAO, april 2002.
2. J.-R. Abrial. *The B Book - Assigning Programs to Meanings.* Cambridge University Press, August 1996.
3. D. Bert, M.-L. Potet, and N. Stouls. GeneSyst: A Tool to Reason about Behavioral Aspects of B Event Specifications. Application to Security Properties. In *Proceedings of ZB2005 Conference, LNCS 3455*, pages 299–318. Springer-Verlag, 2005.
4. F. Bouquet, B. Legeard, and F. Peureux. A constraint Solver to Animate a B Specification. *Int. Journal on Software Tools for Technology Transfer*, 6, 2004.
5. A. Dardenne, A. v. Lamsweerde, and S. Fickas. Goal-directed Requirements Acquisition. *Science of Computer Programming*, 20:3–50, 1993.
6. Edemoi. EDEMOI project web site. http://www-lsr.imag.fr/EDEMOI/, 2004.
7. A. Idani, Y. Ledru, and D. Bert. Derivation of UML Class Diagrams as Static Views of Formal B Developments. In *Proceedings of the 7th Int. Conference on Formal Engineering Methods (ICFEM'05), LNCS 3785*, pages 37–51. Springer-Verlag, 2005.
8. R. Laleau, S. Vignes, Y. Ledru, M. Lemoine, D. Bert, V. Donzeau-Gouge, C. Dubois, and F. Peureux. Application of Requirements Analysis Techniques to the Analysis of Civil Aviation Security Standards. In *Proceedings of SREP'05 - 13th IEEE Int. Requirements Engineering Conf.*, pages 91–106. IEEE, 2005.
9. B. Legeard, F. Peureux, and M. Utting. Automated Boundary Testing from Z and B. In *Proceedings of FME'02 - Formal Methods Europe, LNCS 2391*, pages 21–40. Springer-Verlag, 2002.