EARLY CHECKING OF SYSML MODELS APPLIED TO PROTOCOLS

Pierre de SAQUI-SANNES, Rob. VINGERHOEDS

ISAE-SUPAERO – University of Toulouse
pdss@isae-supaero.fr

Ludovic APVRILLE

LTCI, TelecomParisTech, Université Paris Saclay
ludovic.apvrille@telecom-paristech.fr

ABSTRACT: *The paper shares an experience in using SysML and the free, open-source software TTool for protocol modelling and communication architecture validation. A dialogue between a pilot and a control tower serves as running example to demonstrate the benefits of complementary model analysis techniques: simulation, model checking, and verification by abstraction. The proposed method may be adapted to other modelling languages and tools.*

KEYWORDS: *Modelling, SysML, Simulation, Formal Verification, TTool, Protocol.*

1 INTRODUCTION

The term "protocol engineering" (Bochmann *et al.*, 2010) was coined four decades ago to denote a set of activities that locate protocol modelling at the heart of communication software design. Protocol engineering pioneered Model-Based Systems Engineering at a time where the MBSE acronym was not yet in use.

Relying on Extended Finite State Machines and process algebra, the protocol engineering community developed its own modelling languages and called them "Formal Description Techniques": Estelle, SDL, and LOTOS. Associated tools took formal descriptions as a reference document for early checking of design errors in the protocol's design trajectory, for automatic generation of executable code, and for test sequences generation.

Functionally similar tools have been developed for UML 2.5. The relevance of UML to protocol engineering has been extensively discussed in (Garduno Barrera and Diaz, 2011) relying on the TAU G2 simulator. (Apvrille et al., 2004) also demonstrated the possibility to apply formal verification to UML models of protocols.

The advent of SysML (OMG, 2017) now questions the applicability of SysML in the same area. To contribute to this discussion, this paper discusses a case study of protocol modelling and communication architecture validation using SysML and the use of a free, open-source SysML Editor software: TTool. TTool includes a diagram editor, a simulator, complementary verification modules (model checking, verification by abstraction), and several code generators.

The paper is organized as follows. Section 2 defines a method associated with SysML and TTool. Section 3 specifies the connection set-up procedure that serves as running example and includes assumptions into the SysML model (Saqui-Sannes and Apvrille, 2016). The requirement capture, analysis and design stages of the life cycle are the subjects of Section 4, 5, and 6, respectively. Section 4 to 6 addresses a nominal system, whereas Section 7 discusses a degraded situation (message loss). Section 8 shows how TTool can be interfaced with the model checker UPPAAL (UPPAAL, 2017). Section 9 surveys related work. Section 10 concludes the paper.

2 METHOD

The System Modelling Language, or SysML for short (OMG, 2017), is a diagrammatic modelling language for systems engineering. The OMG standard (OMG, 2017) defines a wide spectrum language, offering tool manufacturers opportunities for customizing the SysML syntax to handle real-time or other types of systems. It is important to note that the SysML standard defines a notation, but not a way of using it. A real challenge is therefore to convince practitioners to incorporate SysML into the method in use in their development approaches and indeed in their company.

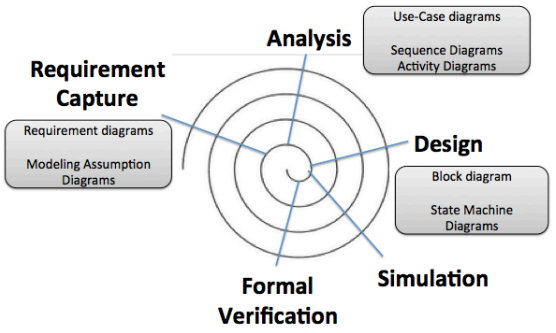

Figure 1: Method and SysML diagrams

As far as protocol engineering is concerned, the method (Figure 1) associated with SysML covers the requirements capture, analysis and design steps of traditional V life cycle. To cope with complexity, the method is further incremental, as suggested by the spiral depicted by Figure 1. Besides the three aforementioned

steps, Figure 1 underlines the importance of simulation and formal verification as two techniques enabling early checking of models against design errors. Figure 1 also indicates the names of the SysML diagrams used by each step in the method.
Requirements capture is the first stage in Figure 1. The "Modelling Assumption Diagram" is not part of the OMG-standard, but is supported by TTool and lists simplifications and other assumptions made at the time of creating the model. Requirement Diagrams, which are part of the SysML standard, define stakeholder, user, and system requirements.

The second stage in the method depicted by Figure 1 is Analysis. A use-case diagram defines the functions and services to be offered by the system and connects these functions or services to the system's environment (actors). Use-cases are documented by scenarios (sequence diagrams) and flow-charts (activity diagrams).

The third stage in the method is design. A block instance diagram defines the architecture of the system. In each block instance lays a finite state machine diagrams defining the behaviour of the block instance.

To make the method in Figure 1 more specific to communication systems, we need to encompass concepts borrowed from the ISO Basic Reference Model (ISO, 2000): layered design, protocol, service, protocol data units and service primitives (Figure 2).

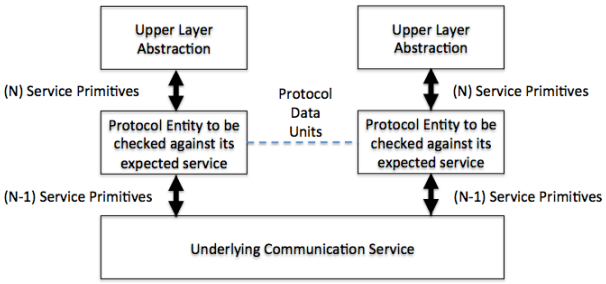

Figure 2: 3-layer architectural pattern

The 3-layer pattern of Figure 1 influences the way we structure the diagrams identified in Figure1.

At requirement capture stage, separate modelling assumptions diagrams are created to address the protocol entities, the underlying network and the user-applications. Similarly, three families of requirement diagrams are created.

At analysis stage, two protocol entities are considered to be the system of interest, whereas the underlying network and the user-applications form the system's environment. Use-cases are documented by means of sequence diagrams. Services scenarios are built up first, considering the system as a black box. Then, they are split up to make the protocol entities appear.

Finally, at the design stage, a 3-layer architecture is defined to comply with the pattern in Figure 2. The architecture is made up of block instances that exchange messages. Of specific interest are the service primitives. The protocol machines are precisely defined. Conversely, the upper layers and the communication service are modelled at a higher degree of abstraction.

3 MODELLING ASSUMPTIONS

This section presents the original specification and lists modelling assumptions. The original specification (Section 3.1) was extracted from the Future Air Navigation System (FANS, 2017) specification and defines a connection set up procedure initiating a dialogue between an aircraft and a control.

3.1 Original Specification

When initiated either by flight crew action or an automatic trigger, an ATS Facilities Notification Contact message is sent and Avionics Timer ATST1 is set. The aircraft AFN application then awaits an AFN Acknowledgement message from ground. Subject to prior agreement with the service provider, the supplementary address field of the ATS Facilities Notification Contact Message may contain an abbreviation of the ATC center address to which the message should be delivered (such as the corresponding IATA or ICAO code). If a successful AFN Acknowledgement message is received within the period of time of ATST1, a positive indication is given to the flight crew and timer ATST1 is cancelled. The AFN acknowledgement message will contain the full 7-character address of the ground AFN end-system. This address should be used in all subsequent AFN messages. If the aircraft AFN receives an AFN acknowledgement message with a non-zero reason code or timer ATST1 expires, an error indication is given to the flight crew.

3.2 Initial Work on the Specification

The specification in Section 3.1 is ambiguous, incomplete and partly inconsistent. Table 1 provides clarifications.

Specification/Problem	Modeling Decisions
The connection set up may be started by the crew or by an automatic trigger.	The pilot will start the connection set up procedure.
Which fields for the Facilities Notification Contact message?	The messages will have no parameter.
After issuing a connection request, the board software sets ATST1 to an undefined value.	ATST1 = 10 minutes.

Table 1 Decisions taken before modelling in SysML

3.3 Modelling Assumptions Diagram

Experience has shown that models are scarcely self-contained and need to be documented to facilitate their sharing and reuse. In particular, a model remains hard to understand for somebody who does not know about the simplifications and more generally the assumptions made by the model's designer. For instance, a protocol developed on top of a pre-existing communication service will be modelled differently depending on whether the service is lossy or not. Consequently, TTool supports Modelling Assumptions Diagrams (MAD) to encourage the model designer to include modelling assumptions into his or her model.

Figures 3, 4 and 5 depict the MAD developed for the case study. The diagrams define tree structures with two types of nodes. The boxes with multiple fields define the assumptions and associate them with attributes that explain the origin of the assumption (*e.g.* the model creator), the status of the assumptions (stating whether it is applied or not in the current version of the model, and the scope indicating, e.g., that an assumption is the consequence of the modelling activity carried out by the designer and not a limitation of the verification tool. Among the relations depicted by Figure 3, let us note the use of <<impact>> to point out how modelling assumptions influences design diagrams. Another important relation is the versioning relation used in Figure 4; it contributes to make the modelling process an incremental one.

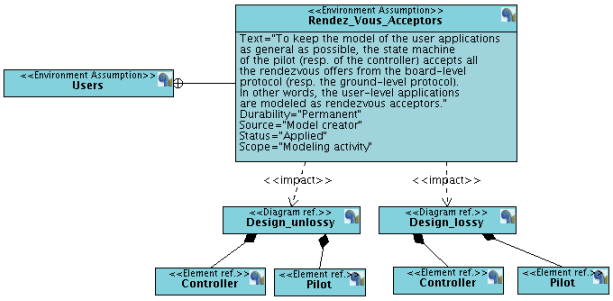

Figure 3: Modelling Assumptions Diagram

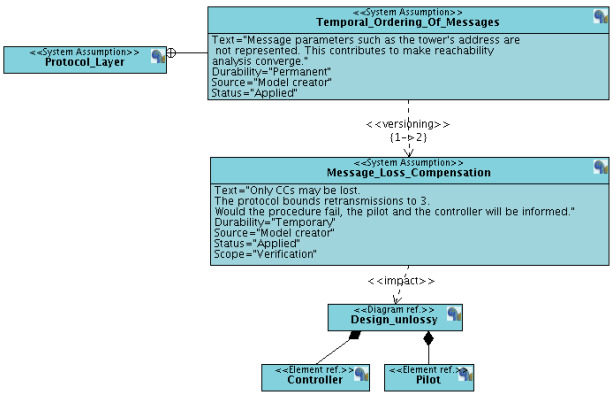

Figure 4: Modelling Assumptions Diagram

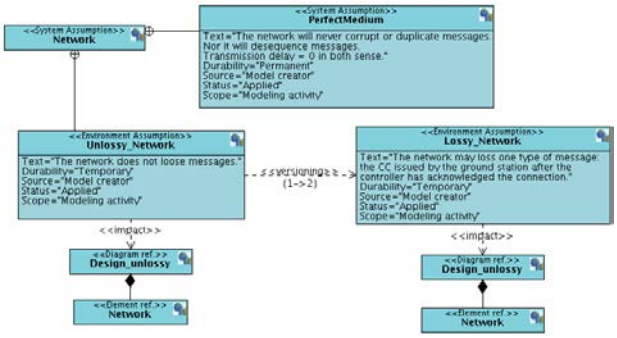

Figure 5: Modelling Assumptions Diagram

4 REQUIREMENTS CAPTURE

The goal of system architecture activities is to define a complete solution based on principles, concepts and properties logically related and consistent with each other. Such solution should have suitable characteristics and properties, matching as well as possible to the problem expressed by a set of system requirements, traceable to mission/business and stakeholder requirements, and traceable throughout life cycle phases and corresponding engineering tools (e.g., mechanical, electronics, software …). This underlines the necessity to obtain pertinent requirements and explains why SysML supports requirement diagrams, a type of diagram not taken on board by UML. As SysML is a language and not a method, there are no constraints to the writing style of requirements. In contrast, an advantage of requirement diagrams is to oblige the system designer to structure and organize the requirements, and to show how this relates to other diagrams in the model.

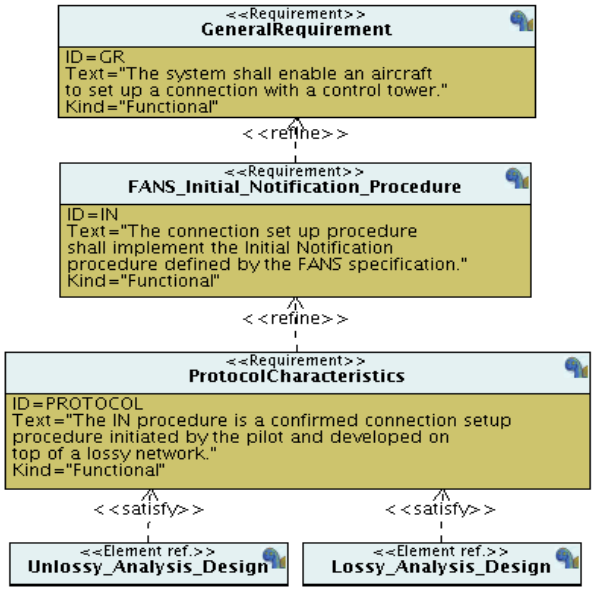

Figure 6: Requirement Diagram

Figure 6 depicts one of the requirement diagram developed for the case study. The tree structure clearly

appears with two types of nodes and two types of arrows. Each node depicted by a box contains one requirement together with its unique identifier, a text, and a categorization between functional and non-functional requirement. The <<refine>> relation from R1 to R2 allows R2 to add more precision to R1. Each <<satisfy>> relation links one design diagram element to one requirement.

5 ANALYSIS

The use-case diagram in Figure 7 defines the boundary of the communicating system. It includes one use-case named `InitialNotification` and connects the latter to three actors that respectively represent the Pilot, the Controller and the pre-existing network the communication system relies on.

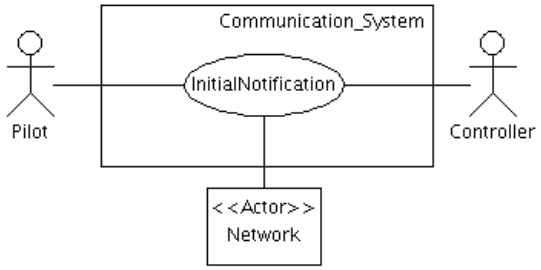

Figure 7: Use-Case Diagram

Two scenarios document the use case. Figure 8 depicts a service scenario where the communication service is a black box and the only messages are services primitives.

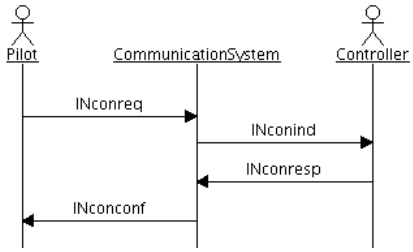

Figure 8: Nominal Case – Service scenario

The protocol scenario in Figure 9 splits up the service scenario, keeping coherence with the latter in terms of service primitives, and adds Protocol Data Units.

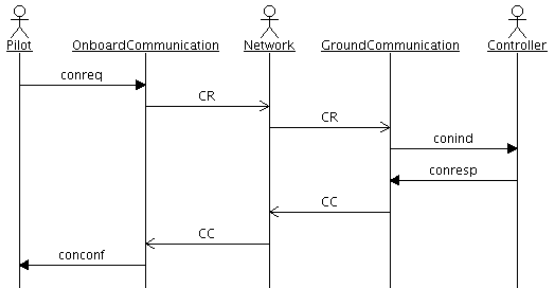

Figure 9: Nominal Case – Protocol Scenario

6 DESIGN

6.1 Architectural Design

The protocol engineering community has adopted the principle of layered design. In practice, two or more protocol entities rely on some pre-existing communication service to render their respective upper-layer user application a value-added service.

Thus, in Figure 10, `OnboardCommunication` and `GroundCommunication` both rely on a communication service modelled by a FIFO queue channel to offer a service to `Pilot` and `Controller`, respectively.

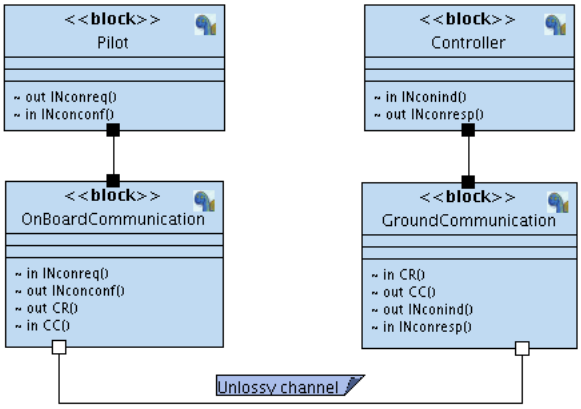

Figure 10: Architecture

Figure 10 depicts an architecture of block instances that communicate via interconnected ports. Black squares denote rendezvous ports that pairs of block instances use for synchronized communication. White square conversely denote FIFO queued communication ports.

6.2 Behavioural Design

Figures 11 to 14 depict the inner workings of the block instances of the architecture in Figure 10. The state machines use three symbols: rounded rectangles modelling states, and message reception / emission.

To keep the `Pilot` as abstract as possible, its state machine accepts all rendezvous from `OnboardCommunication`, either for initiating the procedure with `INconreq` or for successful completion of the connection set up procedure with `INconconf`.

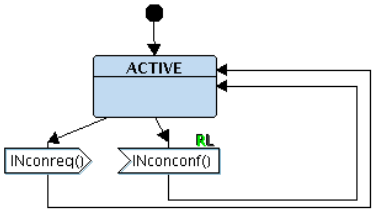

Figure 11: State Machine of the Pilot

The protocol machine of `OnboardCommunication` (Figure 12) enters the `IDLE` state and waits for the `Pilot` to issue a connection request. The protocol machine transforms the latter into `CR`, sends it via the unlossy channel and waits for the remote protocol machine to confirm. Upon reception of a `CC` conveyed by the unlossy channel, the protocol machine sends a confirmation to the `Pilot`.

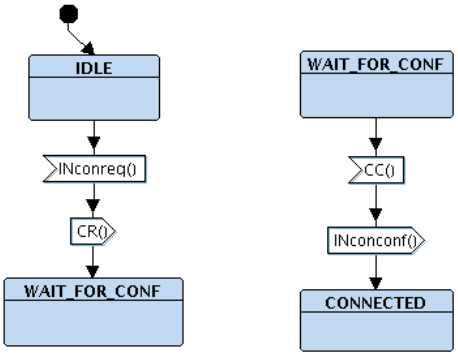

Figure 12: Protocol Machine – On Board Side

The protocol machine of `GroundCommunication` (Figure 13) enters the `IDLE` state and waits for the unlossy channel to convey a CR sent by the remote entity. Upon reception of CR, the protocol machine issues an indication towards the Controller and waits for the latter to response. Upon reception of `INconresp`, the protocol machine issues CC, a confirmation the unlossy channel will convey towards `OnboardCommunication.`

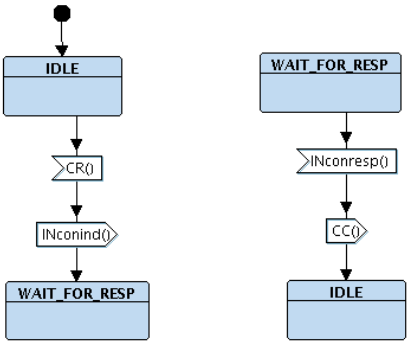

Figure 13: Protocol Machine – Ground Side

Like the state machine of `Pilot`, the one of `Controller` (Figure 14) is a rendezvous acceptor. It handles indications and responses.

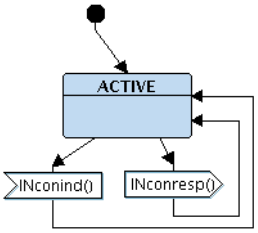

Figure 14: State Machine of the Controller

Unlike requirement capture and analysis that merely generate a set of diagrams, the design phase is not only a matter of drawing: the simulator animates design diagrams and formal verification modules explore the behaviour of the model relying on mathematics rather than chance.

6.3 Model Simulation

TTool is a free and open-source tool that supports several UML profiles, particularly SysML. Figure 15 depicts the main functions on the tool. Note that executable code generation goes beyond the scope of the paper.

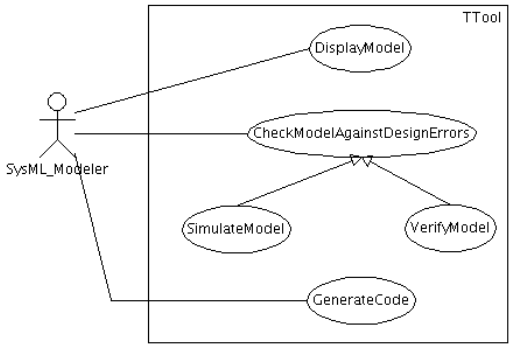

Figure 15: Overview of TTool

The Simulator of TTool (Figure 16) enables animation of state machine diagrams. It takes as input a syntactically- and type-checked SysML model and computes the model's initial global state. Step by step firing of transitions enables early debugging of the model by joint observation of simulation traces in the form of sequence diagrams, annotations on the SysML model itself and display of the state, variables and other elements contained in the blocks the system is made up of. Random firing of transitions enables further exploration of the system's behaviour until a deadlock situation or a termination state is encountered.

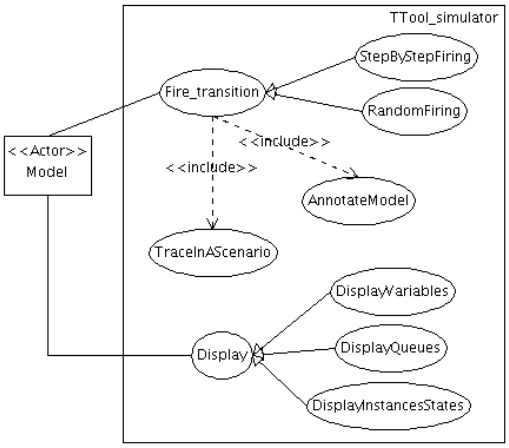

Figure 16: The Simulator of TTool

In Figure 17, the simulation traces takes the form of a sequence diagram and completely depicts the connection procedure.

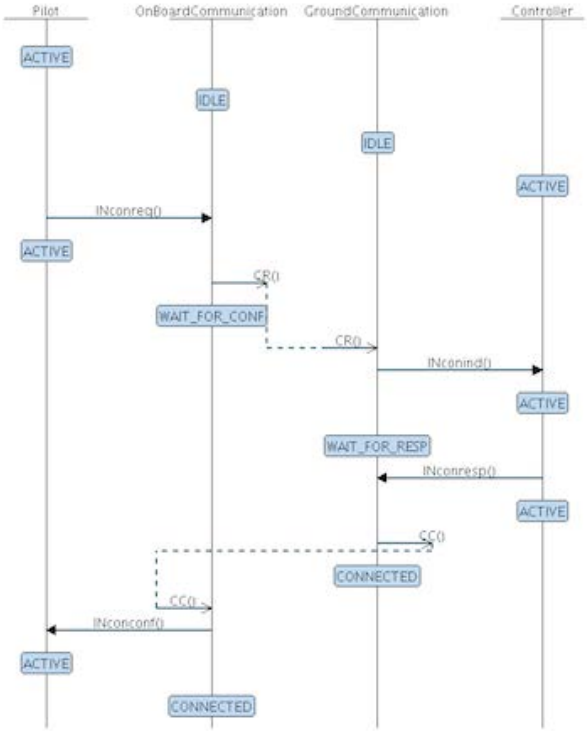

Figure 17: Simulation Trace

6.4 Model Checking

One of the most widespread verification approaches is reachability analysis. Relying on a systematic analysis of the state space of the system under design, reachability analysis may output a so-called "reachability graph" representing all the valid execution paths and states of the system starting from its initial state.

On a general principle, reachability analysis faces the well-known "state explosion problem". Assuming the graph has been computed, the question of exploiting that graph is asked for. An approach that checks whether one property is met or not, and provides a yes/no answer is known as "model checking" (Figure 19). An approach that processes the reachability graph as a Labelled Transition Systems and applies minimization techniques to come up with an abstract view of the system is known as "verification by abstraction" (Figure 18.

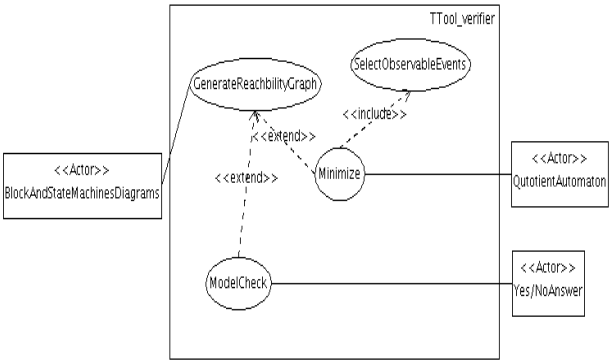

Figure 18: Verification Capabilities of TTool

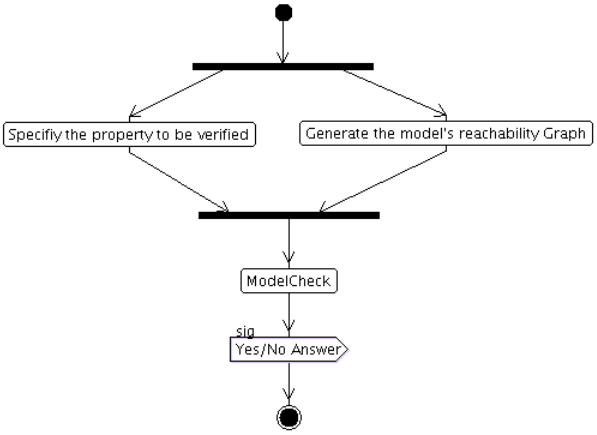

Figure 19: Principles of Model Checking

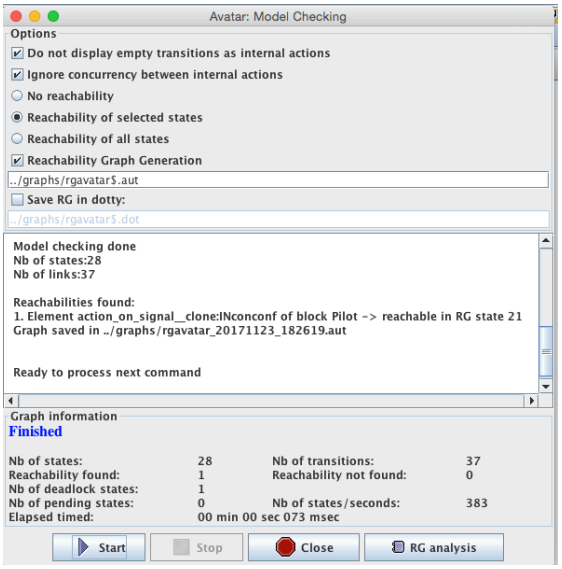

Figure 20: Model-Checker Interface

TTool natively offers a model-checker that does not require expressing a property using a formal language. Any state or event in a state machine may be checked for reachability as soon as it is marked with a RL label (see `INconconf` reception in Figure 11). Figure 20 shows the model-checker interface and the way it states that `INconconf` is reachable.

6.5 Verification by abstraction

The reachability graph (not represented here for space reasons) encounters 28 states and 37 transitions. Labelling the reachability graph with service primitives as observable events (Figure 22) enables application of minimization techniques. Minimization with respect to Milner's observational equivalence outputs the quotient automaton in Figure 23. Thus, from a complete reachability graph that remains hard to explore by hand, verification by abstraction allows one to characterize the service provided by the protocol.

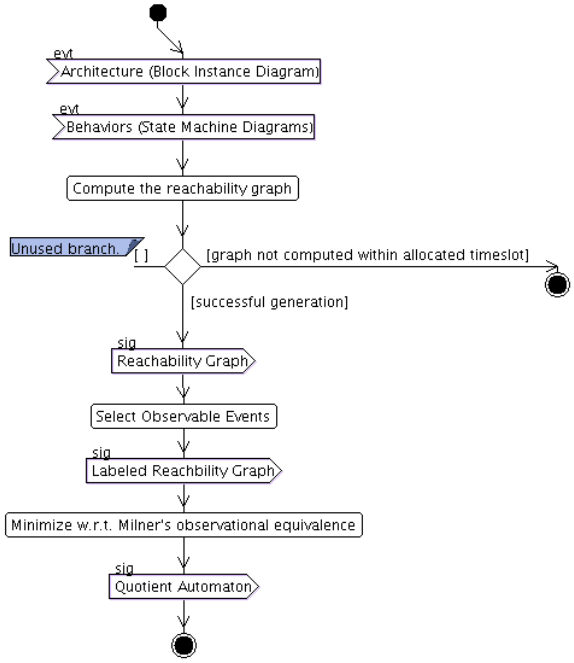

Figure 21: Principles of Verification By Abstraction

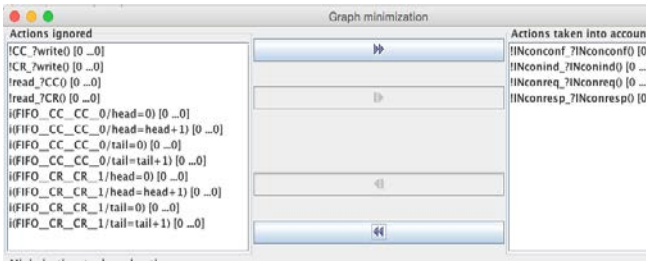

Figure 22: Service Primitives are Made Observable

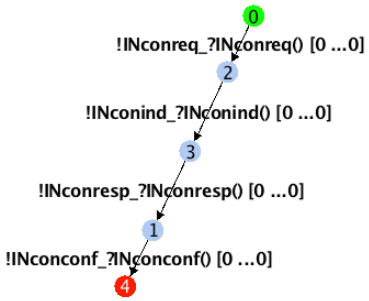

Figure 23: Quotient Automaton

7 DEGRADED CASE

Protocol design is an incremental process. The first version of the protocol machines, as well as of the service primitives list, fits with a perfect underlying communication service that neither losses nor corrupt messages. Version 2 of such model leverages initial restrictive hypotheses and approaches a more realistic version of the model where the underlying communication service may loss one type of message, namely CC.

For space reasons, the paper does not reproduce version 2 of the model entirely. Figure 24 shows how the state machine of the board protocol entity has been extended to handle a (bounded) retransmission counter and to inform the pilot in case of unsuccessful set-up procedure.

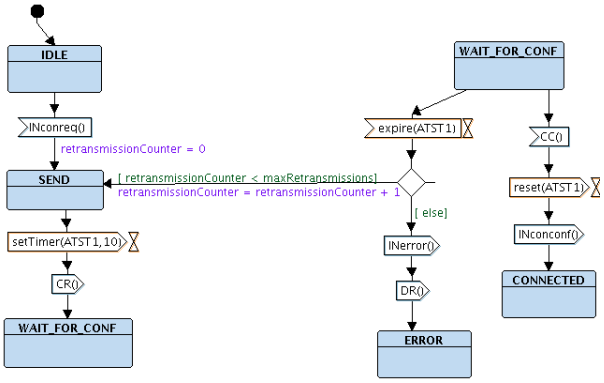

Figure 24: New Protocol Machine On the Ground Side

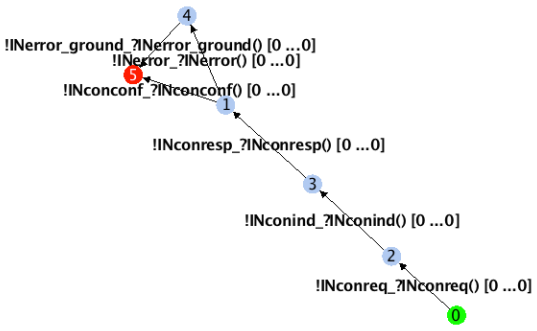

Figure 25: Quotient Automaton

Figure 25 depicts the quotient automaton obtained for the extended SysML model. The triangle distinguishes between successful completion and unset connection.

8 VERIFICATION USING TEMPORAL LOGICS

Safety pragmas can be used in design models in order to capture complex properties expressed in a reduced form of CTL. After checking the syntax of these pragmas, TTool can automatically invoke UPPAAL (UPPAAL, 2017) in order to verify these pragmas.
Pragmas must follow the following format:

- A[] p means that whatever the state of the modelled system, p must be satisfied
- A<> p means that p must be satisfied in at least one state of all possible execution paths
- E[]p means that p must be satisfied in all the states of at least one execution path
- E<>p means that p must be satisfied in at least one state of one execution path.
- p->q means that whenever p is satisfied in an execution path, q will eventually be satisfied in the same execution path.

Figure 26 displays two safety pragmas for the ERROR state in the OnBoardCommunication block. The first pragma is not satisfied, which means that the ERROR state is not reached in all execution paths. Yet, since the second pragma is satisfied, we know that there is at least one execution path that goes through the ERROR state.

Figure 26: Safety pragmas

9 RELATED WORK

Unlike SysML tools that associate formal verification with activity diagrams (*e.g.*, Ouchani et al., 2014) TTool applies model checking to block and state machines diagrams.

TTool also implements verification by abstraction, which, to our knowledge, is not offered by other SysML tools. Let us remark that verification by abstraction was already applied to protocol models expressed in Estelle (Courtiat and de Saqui-Sannes, 1992).

In terms of protocol modelling and communication architecture validation, the book by Garduno and Diaz (Garduno Barrera and Diaz, 2011) discusses a complete case study of Transport-level protocol modelling using UML/SDL, the variant of UML 2 supported by the TAU G2, a tool that offers simulation capabilities similar to the one offered by TTool. Unlike TTool, TAU G2 does not offer verification capabilities.

10 CONCLUSIONS

With their capacity to abstractly model services and protocols, and to check a model against its expected service, SysML and TTool help designers in the early stages of the design trajectory of communicating systems. SysML diagrams and TTool outputs are closed to the notations protocol designers are familiar with. This paper addresses safety issues and concentrates formal verification on the temporal ordering of events. SysML-sec, another language supported by TTool (Li et al, 2017), could be used to detect security flaws. Finally, SysML models are also used to generate test sequences.

REFERENCES

Apvrille, L., Courtiat, J.-P. Lohr, C., de Saqui-Sannes, P., 2014, TURTLE: A Real-Time UML Profile Supported by a Formal Validation Toolkit, IEEE Trans. on Software Engineering, 30 (7), p. 473-487.

Bochmann, G.v., Rayner, D., West, C. H., 2010, Some Notes on the History of Protocol Engineering, Computer Networks, 54(128), p. 3197-3209.

Courtiat, J.-P., de Saqui-Sannes, P, 1992, ESTIM: An Integrated Environment for the Simulation and Verification of OSI Protocols Specified in Estelle, Comp. Networks and ISDN Systems, 25(3), p. 83-98.

FANS 2017, Future Air Navigation Systems, https://en.wikipedia.org/wiki/Future_Air_Navigation_System.

Garduno Barrera, D., Diaz, M., 2011, Communicating Systems with UML2: Modeling and Analysis of Network Protocols, June 2011, Wiley-ISTE, 288 p.

Li, L.W., Lugou, F., Apvrille, L., 2017, Security Modeling for Embedded System Design", 4th International Workshop on Graphical Models for Security, Santa Barbara, CA, USA.

ISO, 2000, ISO/IEC 7498-1:1994, Information technology -- Open Systems Interconnection -- Basic Reference Model: The Basic Model, https://www.iso.org/standard/20269.html

OMG, 2017, Systems Modeling Language 1.5, May 2017, http://www.omg.org/spec/SysML/1.5/.

Ouchani, S., Aït Mohamed, O., Debbabi, M., 2014, A formal verification framework for SysML activity diagrams, Expert Systems with Applications, (41) 6, pp. 2713-2728.

Saqui-Sannes, P. de, Apvrille, L., 2016, Making Modeling Assumptions an Explicit Part of Real-Time Systems Models", 8th European Congress on Embedded Real Time Software and Systems (ERTS), Toulouse, France, pp. 27-29

TTool, 2017, http://ttool.telecom-paristech.

UPPAAL, 2017, http://www.uppaal.org/.