Safety of Unmanned Aircraft Systems Facing Multiple Breakdowns

Patrice Carle[1], Christine Choppy[2], Romain Kervarc[1], and Ariane Piel[1,2]

1 **ONERA – The French Aerospace Lab, 91123 Palaiseau, France**
{patrice.carle,romain.kervarc,ariane.piel}@onera.fr
2 **Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS UMR 7030, 93430 Villetaneuse, France**
Christine.Choppy@lipn.univ-paris13.fr

Abstract

This work deals with data analysis issues in an aeronautics context by using a formal framework relying on activity recognition techniques which are applied to the certification and safety analysis processes of Unmanned Aircraft Systems in breakdown situations. In this paper, the behaviour of these systems is modelled, simulated and studied in case of multiple failures using a complex event processing language called chronicles to describe which combinations of events in time may lead to safety breaches, and a C++ chronicle recognition library is used to implement this method.

1998 ACM Subject Classification I.2.4 Knowledge Representation Formalisms and Methods

Keywords and phrases complex event processing, safety, aeronautics, multiple breakdowns, behaviour recognition tool

Digital Object Identifier 10.4230/OASIcs.FSFMA.2013.86

1 Introduction

The wide range of the possible civil applications to the insertion of aircrafts without pilots on board in controlled or uncontrolled airspace motivates a pronounced general will for its achievement in a near future. One of the main security issues to be solved is the global consistency of the system required to operate safely an Unmanned Aircraft (UA). In the framework of operation safety analysis, we provide the possibility to detect incoherent states between the different entities making up the system. These incoherent states are formalised so as to be able to automatically recognise them through complex event processing, and hence offer the opportunity of both a self-acting surveillance and an assistance to the improvement of the system. This work relies on a fragment of the IDEAS project in charge of the Insertion of Unmanned Aircrafts in Airspace and Security, and tackles consistency problems in breakdown handling policies for UA.

The system required to safely operate an unmanned aircraft can follow several types of architecture. Our model is based on the one presented in Fig. 1. It is composed of three entities, the UA and the Remote Pilot Station (RPS), which both make up the Unmanned Aircraft System (UAS), to which is added the Air Traffic Control (ATC). All three interact via several communication links. The RPS pilots the UA via Telecommand (TC), and the UA sends information to the pilot through Telemetry (TM). In addition, the ATC and the pilot can communicate via radio (Voice) relayed by the UA.

Hence, the dynamic data flows between the agents of the system and between different systems if several UAS are considered are very elaborate. Moreover, each agent deduces

1st French Singaporean Workshop on Formal Methods and Applications 2013 (FSFMA'13).
Editors: Christine Choppy and Jun Sun; pp. 86–91

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

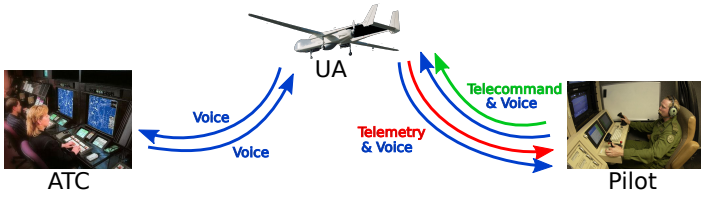

Figure 1 Architecture of the system required to operate safely an UA.

from its own observations the state of the other agents. The situation in case of a fault can therefore be very complex, and all the more so if several faults are considered. Hence, these highly automated systems are very critical, which requires the strong risk-free guarantees provided by formal methods such as our behaviour recognition technique.

In the framework of the IDEAS project, the behaviour of each entity in case of a failure has been specified [5]. In a previous paper [3], we put forward ongoing work overseeing the consistency between the three entities during the rundown urgency procedure linked to a single Telecommand (TC) failure (the pilot receives information from the UA via telemetry but cannot send out orders to it). In the present paper, we will consider the additional failure of the Voice link (the pilot and the ATC cannot communicate directly anymore), both on its own and coupled with the TC breakdown, providing a framework for an automatic monitoring of consistency in UAS. This yields an opportunity to put to use new features of the monitoring tool employed for the online analysis of the simulation.

The problem is formalised by standardising the rundown procedures from the IDEAS project into the UML language [6]. A class diagram exposed in [3] precisely describes the structure of the system. The three entities are made up of several components modelling their complex interactions precisely enough to be able to consider separately the smooth functioning of the communication links, and hence check for the consistency between the active states of the diagram. The several urgency procedures corresponding to the codified behaviours related to failures and specified in the IDEAS project are united and translated into one single state-transition diagram presented in Fig. 2. This allows to consider multiple concurrent failures which was not possible before.

This paper will start by going into some details of the preliminary semi-formal representation of the system, exposing the stakes at hand. In a second part, the system will be supervised using a behaviour recognition library, and the results will be displayed.

2 A Modelling framework

Through the class diagram and the state-transition diagram, the system has been entirely modelled. Indeed, its life cycle is mirrored by the changes in the active states of the diagram (Fig. 2). The aim is to analyse the simulation while it is running in order to draw forth certain desired or undesired behaviours.

The simulation which has been established strictly follows the behavioural guidelines specified and required by the IDEAS project. However, UA are not currently allowed to fly in controlled or uncontrolled airspace, and the regulations such as the ones studied in this paper have not yet been finalised. Our aim is to study whether certain incoherent states emerge from the simulation. Not only does behaviour recognition on running simulation allow to confirm or reject certain requirement choices, but it also provides a means to enlighten the experts in their elaboration of the regulations that should be followed, and this by bringing out and highlighting possible security breaches. In this application, the first use of our

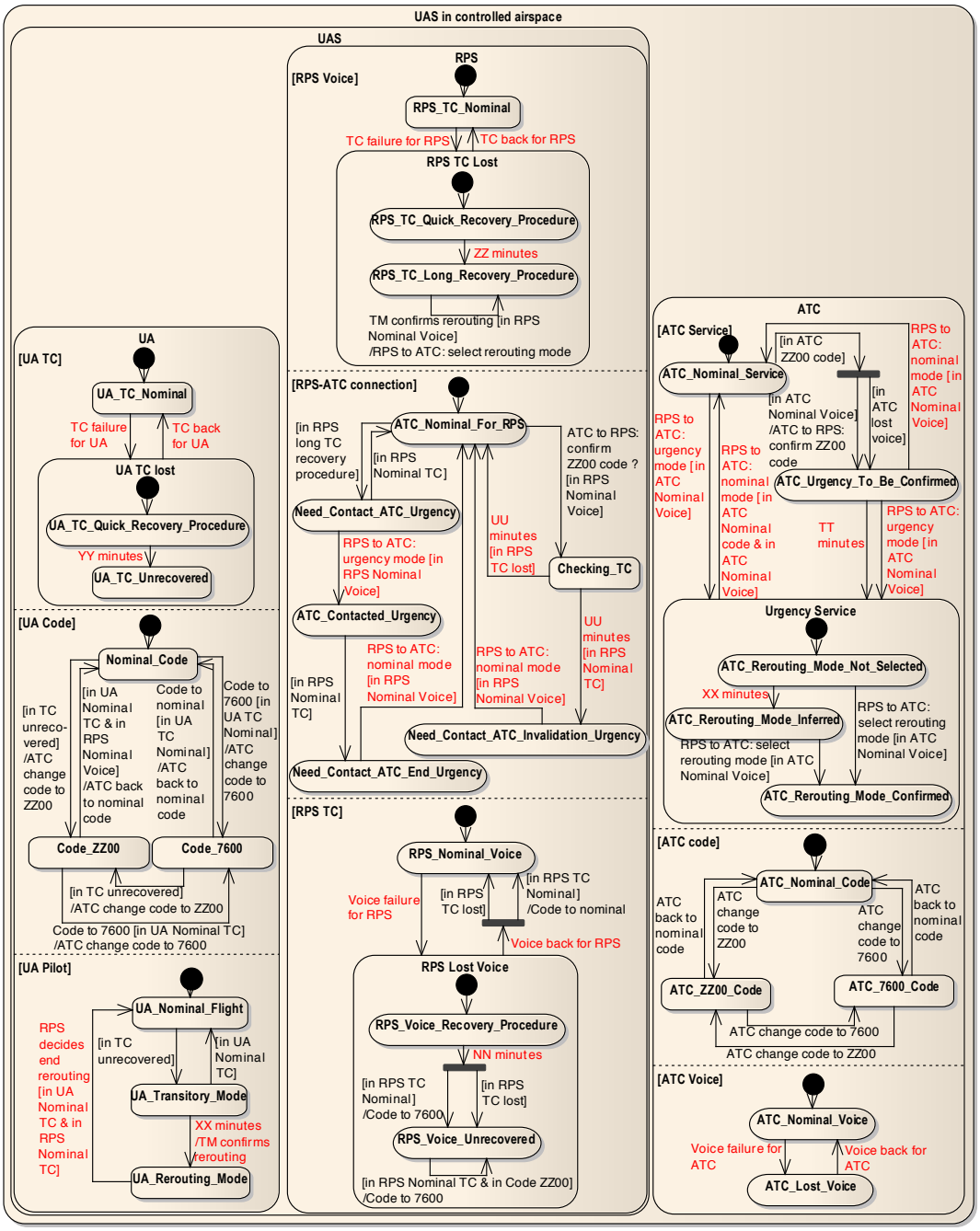

Figure 2 State-transition diagram describing the behaviour of the system.

behaviour recognition technique is to offer an assistance during the development stage of the regulations. Since multiple breakdowns are considered, the situation becomes highly complex to embrace, and such assistance is necessary.

Eventually, the remaining causes of safety breaches should only be human (i.e. due to the pilot or the air traffic controller). These have to stay in the model since they represent a reality which cannot be avoided, but they can be detected using our behaviour recognition technique. Hence, the second use of our method is the detection of the last safety breaches which cannot be prevented, in order to generate alarms and reduce the potential risks.

The first step is to specify the inconsistent states of the system which have to be averted. For example, the two following behaviours are undesired:

- *Incoherent ATC Voice*: the transponder code emitted by the UA starts indicating code 7600 at the air traffic control, which means that there is a voice failure, but the controller does not realise so, and this is expressed by the fact that the diagram does not switch to `ATC Lost Voice`.
- *Incoherent flight mode UA/ATC*: after a fault which has been solved, the UA has switched back to a nominal flight but the ATC stays in an urgency service.

Once one of these behaviours is detected, its origins have to be determined. If the cause is due to faulty behavioural guidelines, then the model has to be corrected, and, otherwise, if the source is human, it should be planned to trigger alarms warning the pilot and/or the air traffic controller of the situation.

In order to exploit these examples and to allow direct simulation, the UML diagram has been implemented in C++ using the Meta State Machine (MSM) library [4] of boost (Version 1.53.0) which provides a straightforward way to define state machines. So as to simulate the life cycle of the system, scenarios activate the red transitions of the diagram of Fig. 2 (the other transitions are triggered by events automatically generated by the diagram), thus providing a complete modelling framework. We thus obtain a direct simulation of the system that we want to supervise using failure detection.

3 Behaviour recognition with CRL (Chronicle Recognition Library)

To perform this supervision, it is necessary to be able to formally express failures as behaviours which are to be recognised. Monitoring is then needed to allow an *online* recognition of *all* the occurrences of the described behaviours during the running simulation, which are central issues linked to complex event processing. These two requirements are fulfilled by a temporal language — the chronicle language, which syntax and semantics are partly defined in [2] — and its associated recognition tool.

This language allows to formally depict system behaviours. Arrangements of events are described: a chronicle can be a single event, the conjunction of two chronicles, the disjunction of two chronicles, the sequence of two chronicles or the absence of a chronicle during another chronicle. In addition, temporal constraints between chronicles or on the length of time of the recognition of a chronicle may be specified. For instance, let `E` and `F` be single events and δ a real number, chronicle (`E` then δ) $-$ [`F`] corresponds to event `E` followed by δ units of time during which no event `F` occurs.

The first step is therefore to write down chronicles which will oversee the system. It is necessary to take into account isolated events since we want to be able to recognise the chronicles online, so the set of events considered to build these chronicles are the entrances in and exits from the different possibly active states of the diagram of Fig. 2.

The two unwanted behaviours briefly described in Sec. 2 may be formally expressed by the following chronicles:

- *Incoherent ATC Voice*

  (`to_ATC_Nominal_Code to_ATC_7600_Code` then 5) $-$ [`to_ATC_Lost_Voice`]
- *Incoherent flight mode UA/ATC*

  (`from_UA_Nominal_Flight`
  ((`to_UA_Nominal_Flight` then 10) $-$ [`from_UA_Nominal_Flight`]))
  $-$[`to_ATC_Nominal_Service`]

Once the behaviours to be recognised have been formalised in the chronicle language, the simulation is run along different scenarios which are supervised by a behaviour recognition tool in charge of bringing to light any incoherent state described by the written chronicles. Such a tool, designed on the basis of duplicating automata and called Chronicle Recognition System (CRS/ONERA), has been developed by the ONERA in the late 1990s [1]. A new recognition tool which algorithms directly result from the set semantics of the chronicle language is developed during this Ph.D. thesis. This tool, called Chronicle Recognition Library (CRL), implemented in C++, is described in greater detail in [3]. Chronicles are plugged into the program, and then, gradually as events flow in, the program gives the set of all the recognitions of each chronicle, specifying for each recognition which events lead to it.

Let us now run this tool on two scenarios, looking for recognitions of the two previously specified chronicles.

Consider, to start with, an overly simple story line as an instructive example: there is a voice failure, which is only acknowledged by the pilot (event `Voice failure for RPS`). The simulation is run with this single event. The evolution of the entrances in and exits from the active states of the diagram are then plugged into CRL which generates the following result:

```
t = 0  Engine created
t = 0  Added chronicle :
        ([(to_ATC_Nominal_Code to_ATC_7600_Code) + 5] − to_ATC_Lost_Voice)
t = 0  Added Event : Voice_failure_for_RPS
t = 0  Added Event : from_RPS_Nominal_Voice
t = 0  Added Event : to_RPS_Voice_Recovery_Procedure
t = 4  Added Event : from_RPS_Voice_Recovery_Procedure
t = 4  Added Event : to_RPS_Voice_Unrecovered
t = 4  Added Event : from_Nominal_Code
t = 4  Added Event : to_Code_7600
t = 4  Added Event : from_ATC_Nominal_Code
t = 4  Added Event : to_ATC_7600_Code
t = 9  Chronicle recognition :
        ([(to_ATC_Nominal_Code to_ATC_7600_Code) + 5] − to_ATC_Lost_Voice)
        Reco Set = {⟨⟨(to_ATC_Nominal_Code, 0), (to_ATC_7600_Code, 4)⟩, (t, 9)⟩}
```

Chronicle *Incoherent ATC Voice* has been recognised: `to_ATC_Nominal_Code` at time 0 has been followed by `to_ATC_7600_Code` at time 4, and, until time 9, the forbidden events have not occurred. Thanks to event historisation, it can be diagnosed that the source of the inconsistency is a lack of attention from the air traffic controller. An alarm should be triggered by the chronicle to warn the ATC of the situation and attempt to restore a correct situation.

Let us now consider a second scenario involving multiple breakdowns: a voice failure acknowledged both by the pilot and the ATC (events `Voice failure for RPS` and `Voice failure for ATC`) is shortly followed by a TC failure recognised both by the UA and the pilot (events `TC failure for UA` and `TC failure for RPS`). However, the TC is restored 15 minutes later (events `TC back for UA` and `TC back for RPS`), at which point the pilot decides that the situation is not too alarming (a voice failure can indeed be considered as such) and therefore orders the UA back to a nominal flight (event `RPS decides end rerouting`). When the simulation is run with these events, the following result is identified by CRL:

```
⋮
t = 65  Chronicle recognition :
            ([from_UA_Nominal_Flight ([to_UA_Nominal_Flight + 10]
            − from_UA_Nominal_Flight)] − to_ATC_Nominal_Service)
            Reco Set = {⟨(from_UA_Nominal_Flight, 35), ⟨(to_UA_Nominal_Flight, 55), (t, 65)⟩⟩}
```

Chronicle *Incoherent flight mode UA/ATC* is recognised. This time, the inconsistency is not due to human error, which means that the model has to be corrected: it is brought to light that a transition is missing in the modelling of the ATC between `Urgency service` and `ATC_Nominal_Service`. Indeed, the ATC should be able to switch back to a nominal service even though no radio communication with the pilot is available. A transition triggered by the exit of `ATC_ZZ00_Code` (indicating the end of the TC failure) has therefore to be added. Once this improvement has been completed and the system and/or the procedure have been modified, running the simulation using the new model on the same scenario does not produce any recognition anymore, ascertaining that the behaviour has been rightfully corrected.

4 Conclusion and perspectives

In conclusion, we provide in this paper a complex event processing framework applied to monitor safety for Unmanned Aircraft Systems in case of one or multiple breakdowns. The behaviour of the UAS is completely modelled in a UML diagram which is implemented in C++. A temporal language, the chronicle language, is used to specify the inconsistent states which would lead to safety breaches and which therefore have to be avoided. CRL, a C++ library, allows direct analysis of simulation data. The achievement is twofold: an assistance to regulation development is provided, and the remaining safety breaches which cannot be totally prevented can be made to trigger alarms.

Among the numerous possible future directions for this work, we plan to continue the extension of the chronicle language in order to increase its expressivity and hence be able to deal with a wider spectrum of applications. For instance, we intend to formalise a notion of actions triggered by successful recognitions which would allow, for example, the formalisation of the alarm generation in the application presented in this paper. In addition, the choice and writing of the chronicles to be recognised is currently completed by an expert by hand. It would be desirable to develop an assistance tool for the generation of chronicles, so as to get closer to an exhaustive covering of the situations to be recognised.

Acknowledgements. The authors thank J. Bourrely for his help and support for this application, and T. Lang and C. Le Tallec for their useful insights on Unmanned Aircrafts.

References

1 P. Carle, P. Benhamou, F.-X. Dolbeau, and M. Ornato. La reconnaissance d'intentions comme dynamique des organisations. In *6èmes Journées Francophones pour l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents (JFIADSMA'98)*, 1998.

2 Patrice Carle, Christine Choppy, and Romain Kervarc. Behaviour recognition using chronicles. In *Proc. 5th IEEE International Symposium on Theoretical Aspects of Software Engineering*, pages 100–107, 2011.

3 Patrice Carle, Christine Choppy, Romain Kervarc, and Ariane Piel. Handling Breakdowns in Unmanned Aircraft Systems. In *18th International Symposium on Formal Methods - Doctoral Symposium*, 2012.

4 Christophe Henry. "MSM library of boost". `www.boost.org/doc/libs/1_48_0/libs/msm/doc/HTML/index.html`, 2011.

5 Thibault Lang. IDEAS–T1.1: Architecture de système de drone et scénarios de missions. Technical report, 2009.

6 "OMG Unified Modeling Language™(OMG UML), Superstructure, Version 2.4.1", 2011.