Model Checking and the Certification of Autonomous Unmanned Aircraft Systems

Matt Webster[1], Michael Fisher[2], Neil Cameron[1] and Mike Jump[1,3]

[1] Virtual Engineering Centre, Daresbury Laboratory, Warrington, UK
[2] Department of Computer Science, University of Liverpool, UK
[3] School of Engineering, University of Liverpool, UK

{matt,mfisher,ncameron,mjump1}@liverpool.ac.uk

Abstract

In this paper we assess the feasibility of using formal methods, and model checking in particular, within the certification of Unmanned Aircraft Systems (UAS) for civil airspace. We begin by modelling a basic UAS control system in PROMELA, and verify it against a selected subset of the CAA's Rules of the Air using the SPIN model checker. We then refine our UAS control system to incorporate probabilistic aspects, verifying it against the same Rules of the Air using the probabilistic model checker PRISM. This shows how we can measure statistical adherence to such rules. Next we build a similar UAS control system using the autonomous agent language Gwendolen, and verify it against the small subset of the Rules of the Air using the agent model checker AJPF. We introduce more advanced autonomy into the UAS agent and show that this too can be verified. Finally we compare and contrast the various approaches, discuss the paths towards full certification, and present directions for future research.

1 Introduction

An Unmanned Aircraft System (UAS, plural UAS) is a group of individual elements necessary to enable the autonomous flight of at least one Unmanned Air Vehicle (UAV) [8]. For example, a particular UAS may comprise a UAV, a communication link to a ground-based pilot station and launch-and-recovery systems for the UAV.

The perceived main advantages of UAS in military applications come from their ability to be used in the so-called dull, dangerous and dirty missions, e.g., long duration flights and flights into hostile or hazardous areas (such as clouds of radioactive material) [18]. There is a growing acceptance that the coming decades will see the integration of UAS into civil airspace for a variety of similar applications: security surveillance, motorway patrols, law enforcement support, etc. [19, 14]. However, in order for this integration to take place in a meaningful way, UAS must be capable of routinely flying through "non-segregated" airspace (as opposed to segregated airspace which is for the exclusive use of specific users). Typically today, for civil applications, UAS fly in segregated airspace. This is not an acceptable solution if the demand for UAS usage increases as is envisaged. The UK projects ASTRAEA and ASTRAEA II and the FAA's Unmanned Aircraft Program Office (UAPO) are tasked with this objective but a summary of the issues is considered pertinent. Guidance on the UK policy for operating UAS is given in [8]. The overarching principle is that, "UAS operating in the UK must meet at least the same safety and operational standards as manned aircraft". A UAS manufacturer must therefore provide evidence to the relevant regulatory authority that this is the case.

For manned aircraft, there is a well understood route for manufacturers to demonstrate that their vehicle and its component systems meet the relevant safety standards (see, for example, [11]). However, the manufacturer does not have to concern itself with certification of the pilot: it is assumed that a suitably qualified crew will operate the aircraft. For a UAS, however, the human operator may be out of the control loop (for example, either due to a lost data link or because of the nature of the system designed) and therefore the manufacturer must demonstrate that any autonomous capabilities of the aircraft, in lieu of an on-board human pilot, do not compromise the safety of the aircraft or other airspace users. The acceptable means to achieve this end, i.e., regulatory requirements, have yet to be formalised even by the regulators.

In this paper, therefore, we investigate the potential usefulness of model checking in providing formal evidence for the certification of UAS. The work described in this paper is a study examining the feasibility of using formal methods tools to prove compliance of an autonomous UAS control system with respect to a small subset of the "Rules of the Air" [7]. Demonstrating that the decisions made by the autonomous UAS are consistent with those that would be made by a human pilot (presumably in accordance with the Rules of the Air), would provide powerful evidence to a regulator that the UAS would not compromise the safety of other airspace users. Thus, the work described herein helps to answer the question as to whether or not formal verification tools have the potential to be able to meet or contribute to this overall ambition.

This is one step on the way towards the certification of autonomous UAS in non-segregated UK airspace. However, this study allows us to show how a route to full certification might be relieved of some of the burden of analysis/testing required by the current regulatory framework. This might save time and increase reliability, but might come at the cost of a required increase in the level of expertise required of the analysts involved in the certification process.

1.1 Approach

Since the route to airframe and automatic control system certification is already established, the main, and possibly the only, difference between a UAS and a human-piloted aircraft is the core autonomous control system, plus all of the systems that are directly associated with it e.g., power supplies, etc. Thus, a vital part of certification is to show that this core autonomous control (in the form of an "intelligent" agent) would make the same decisions as a human pilot/controller would make (this is, after all, one of the piloting skills that a human pilot must obtain to be awarded a licence).

In general, analysing human behaviour in this way is, of course, very difficult. However, in the specific case of aircraft certification, pilots should abide by the Rules of the Air. Thus, our approach here is to verify that all of the choices that the agent makes conform to these Rules of the Air. To show how this could be done, we chose a small subset of the Rules of the Air ("The Rules of the Air Regulations 2007," is large, around 50 pages of text, see [7]) and encoded these in a basic temporal logic. Since the rules are, at first glance, fairly straightforward we started with simple model checking using SPIN to show that a sample UAS agent satisfied the selected subset.

Since a real UAS will surely involve uncertain interactions (e.g., errors in sensing) we next extended the scenario to also cover probabilistic aspects of the agent's environment, such as an unreliable sensor. Then we used PRISM [16] to carry out the required probabilistic verification to discover the effects on the agent's behaviour.

Finally, we used a rational agent model. This involved two aspects. The first was to allow more "intelligence" in the UAS agent itself. This extended the agent's choices to take into account not only the situation but also the agent's beliefs about the intentions of other UAS/aircraft. We then used our agent model checking system (AJPF [2]) to verify that this, more "intelligent", agent still satisfied the Rules of the Air. The final aspect was to consider more than the literal meaning of the "Rules of the Air". Specifically, we noticed that there is often an implicit assumption within these rules. For example, "in situation A do B" might have an implicit assumption that the pilot will assess whether doing B in this particular situation would be dangerous or not. Really such rules should be: "in situation A do B, unless the UAS believes that doing B will be likely to lead to some serious problem". In piloting parlance, the agent needs to demonstrate *airmanship*. Thus, the last part of the work was to show how we might "tease" out such aspects into formal specifications involving intentions/beliefs that could then be checked through our verification system.

Thus, we begin by:

1. describing an *abstraction* of the UAS control system — typically, this will comprise (a fragment of) the high-level design for the software;

2. *selecting* a small subset of the Rules of the Air to consider and formalise

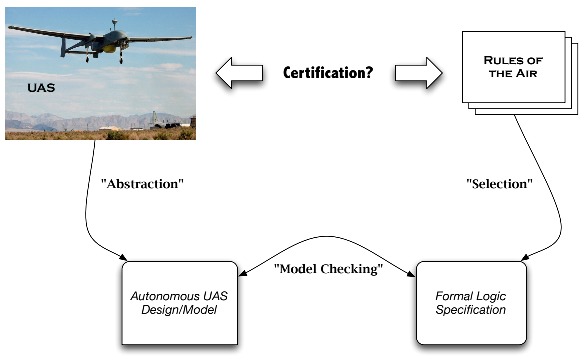

Figure 1: An approach to UAS certification regarding the Rules of the Air.

these within our logical framework; and

3. utilising model checking [9] to automatically check that all possible routes through the design (1) satisfy our logical requirements (2).

Clearly, the closer the UAS design/model is to the actual UAS control system implementation and the closer the logical specification is to the actual *meaning* of the "Rules of the Air", the more useful that model checking will be in generating analytical evidence for certification. Ideally, the UAS model/design should be a description of all the decisions/choices the UAS can possibly make. For the purposes of this study, we assume that standard V&V techniques for high integrity software have been used to ensure that the UAS control system does actually correspond to this design/model.

Ideally, we would also like to capture *all* the Rules of the Air in a precise, logical form. However, there are several problems with this. First, the Rules of the Air are neither precise nor unambiguous — thus it is very hard to formalise their *exact* meaning without making the formulation very large. Next, the number of rules is too large to tackle them all in this small study. Finally, some of the rules implicitly use quite complex notions, such as "likelihood", "knowledge", "the other pilot's intention", "expectation", and so on. While extending our formalisation to such aspects will be tackled in the second half of this study, our initial step is to *select* a small number of rules that are clear, unambiguous, and relevant to UAS.

Our approach is summarised in Figure 1.

1.2 Concepts and Terminology

The UAS model/design will be described as an *executable agent model*, initially using PROMELA [12], but later in higher-level agent languages [1] or lower-level probabilistic state machine languages [16]. The concept of an "agent" is a popular and widespread one, allowing us to capture the core aspects of autonomous systems making informed and rational decisions [23]. Indeed, such agents are typically at the heart of the hybrid control systems prevalent within UAS. We will say more about the “agent” concept later but, for the moment we simply equate “agent” with “process”. Thus, we model the UAS’s choices/decisions as a single process, initially in PROMELA.

1.3 Selecting Rules of the Air for Model Checking

We chose a small subset of three Rules of the Air which were relevant for a straightforward flight of a powered UAS vehicle (e.g., taxiing to take-off, navigation, sense-and-avoid and landing). It was also desirable to choose rules which may potentially be in conflict, as this would present a greater challenge for engineering and verification of the UAS. We also had to leave out certain rules concerning specific heights and distances, as we did not intend to model such detailed information within our UAS model. In addition we wanted to focus on two key scenarios for UAS engineering: (i) “sense-and-avoid”, where the UAS must detect objects that it may collide with and take evasive action; and (ii) partial autonomy, where the UAS proceeds autonomously but checks with a human for permission to perform certain actions. Both are essential abilities of autonomous UAS [19]. Thus, the rules chosen were as follows:

1. **Sense and Avoid**: “. . . when two aircraft are approaching head-on, or approximately so, in the air and there is danger of collision, each shall alter its course to the right.” (Section 2.4.10)

2. **Navigation in Aerodrome Airspace**: “[An aircraft in the vicinity of an aerodrome must] make all turns to the left unless [told otherwise].” (Section 2.4.12(1)(b))

3. **Air Traffic Control (ATC) Clearance**: “An aircraft shall not taxi on the apron or the manoeuvring area of an aerodrome without [permission].” (Section 2.7.40)

The first rule is relevant for the sense-and-avoid scenarios (see (i) above), and the third rule is relevant for partial autonomy (see (ii) above). The second rule is interesting because it may conflict with the first rule under certain circumstances, e.g., where an object is approaching head-on and the UAS has decided to make a turn. In this case, the UAS vehicle may turn left or right depending on which rule it chooses to obey.

Some simplification was necessary to encode the Rules so that they could be model checked. For instance, in the first rule, there are number of factors which “tell” the UAS vehicle to make a turn to the right, such as the pattern of traffic at an aerodrome, ground signals or an air traffic controller. We chose to model all of these under the umbrella term “told otherwise”, as we did not intend to model these factors separately.

1.4 Paper Structure

In Section 2 we describe a model of a basic UAS control system in PROMELA, and in Section 2.2 verify it against a small subset of the Rules of the Air using the SPIN model checker. In Section 3 we refine the UAS control system to incorporate probabilistic aspects and verify it against the same Rules of the Air using the probabilistic model checker PRISM. We show that it is possible to calculate statistical measures of the level of adherence to a Rule of the Air using PRISM's probabilistic features. In Section 4 we construct a basic UAS control system using the autonomous agent language Gwendolen [10], and show that it can be verified against the same Rules of the Air using the agent model checker AJPF [2]. We introduce advanced autonomous behaviour into the UAS agent, and show that this can also be verified as in accordance with the small subset of the Rules of the Air. Finally, in Section 5 we compare the different approaches to UAS agent modelling and verification, and we present directions for future research.

2 Reactive UAS Agents

An agent is a computer system which is situated in an environment and is capable of autonomous action within its environment in order to achieve its objectives [23]. Agents provide a natural way of describing core autonomy, and are therefore a natural way of describing hybrid control systems for autonomous UAS.

Through consultations with researchers from the Autonomous Systems Research Group at BAE Systems (Warton) we have modelled fragments of a typical UAS agent relevant to our selected scenario. Here, it is assumed that the UAS agent will be composed of a set of rules concerning the successful completion of the mission and the safe flight of the aircraft. Each rule has a condition which must be satisfied for that rule to be applied, and a consequence of applying that rule. For example, a rule might look like:

```
IF aircraft_approaching_head_on THEN turn_right
```

This would be the part of the agent designed to deal with the "approaching head-on" scenario described in Section 1.3. There would presumably be many other rules in the agent to deal with other situations, such as running low on fuel, take off, landing, etc. The idea is that the complete set of rules would enable the flight of the UAS, so that the UAS would respond appropriately in every situation. Another such rule could be:

```
IF ATC_clearance_rcvd
THEN set_flight_phase_taxi; taxi_to_runway_and_wait
```

This rule would specify that when the UAS receives clearance from ATC, it will set its flight phase to "taxi" and start taxiing to the runway where it will wait for

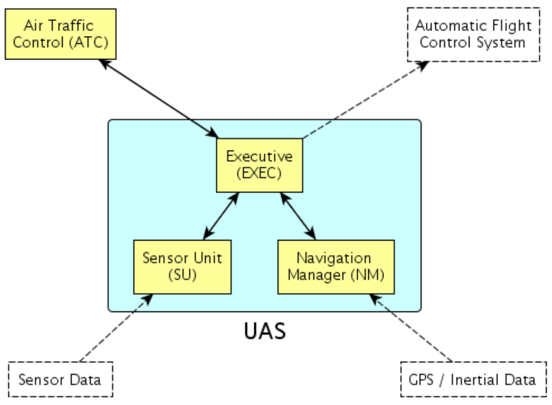

Figure 2: A Model of a UAS Architecture. Boxes with unbroken borders represent processes in the PROMELA model. Boxes with broken borders represent aspects of a real-life UAS system that are not modelled explicitly. Unbroken arrows represent information flow between processes/aspects.

take-off clearance. In general, this kind of agent is known as a *reactive agent*, as it reacts to situations without reasoning about them. (In later sections we will also consider a *practical reasoning*, or *rational*, agent for a UAS.)

2.1 Modelling a Reactive UAS Agent in PROMELA

A simple model of a partial UAS control system has been written using PROMELA, the process modelling language for the SPIN model checker [12]. The UAS is divided up into a number of components: the Executive, the Sensor Unit (SU) and the Navigation Manager (NM) (see Figure 2).

The role of the Executive is to direct the flight of the UAS based on information it receives about the environment from the SU and the NM. The NM is an independent autonomous software entity (i.e., an agent) on-board the UAS which detects when the UAS is off-course and needs to change its heading; it sends messages to the Executive to this effect. When the UAS's heading is correct, the NM tells the Executive so that it can maintain its current heading. The SU is another agent on-board the UAS whose job it is to look for potential collisions with other airborne objects. When it senses another aircraft it alerts the Executive. The SU then notifies the Executive when the detected object is no longer a threat.

Another essential part of the model is the ATC. The Executive communicates with the ATC in order to request clearance to taxi on the airfield. The ATC may either grant or deny such clearance. Thus, our simple reactive UAS models sense-and-avoid scenarios as well as navigation and ATC clearance.

In PROMELA, we model the Executive, the SU, the NM and the ATC as processes, which communicate using message-passing *channels* (see Figure 2). For simplicity we specify the NM and the SU as non-deterministic processes which periodically (and arbitrarily) choose to create navigation and sensory alerts. The Executive process has a variable, called `state`, which contains different values to represent the different parts of the UAS's mission: `WaitingAtRamp` (start of mission), `TaxiingToRunwayHoldPosition`, `TakingOff`, `EmergencyAvoid`, etc.

Each step in the process is modelled by a different value of the `state` variable. Once the UAS model becomes "airborne", the Executive may receive messages from both the SU and the NM. If the Executive receives a message from the SU saying that there is an object approaching head-on, then it updates the state to "Emergency Avoid" and alters the course of the UAS to the right (by updating a variable `direction`). When the SU tells the NM that the object approaching head-on has passed, the Executive will continue on the heading and in the state it was in before the alert, e.g., if it was changing heading and turning left then it will go back to this behaviour. At any point the Executive may receive a message from the NM telling it to alter its heading, maintain its current heading or, eventually, land.

Certain elements of a real-life UAS are not modelled here. We do not model the "real world" environment of the UAS explicitly; rather we use the SU to send sensory alerts on a non-deterministic basis. Likewise, the NM does not really navigate, as there is no "real world" in the model to navigate through, and so it sends navigation alerts on a non-deterministic basis. Also, we do not model the flight control systems of the UAS or any aspects of the vehicle itself, as without a "real world" model these are unnecessary. However, we make these simplifications without loss of accuracy in the verification process: our aim is verify the behaviour of the Executive, to ensure that it adheres to the "Rules of the Air" according to the information it possesses about the current situation, and so using the SPIN model checker we can ascertain whether the Executive behaves in the desired manner.

2.2 Model Checking the Rules of the Air in SPIN

Now we have a model of some aspect of the behaviour of a UAS, together with elements of its environment (e.g., ATC) we can check its compliance with the Rules of the Air identified in Section 1.3 using the SPIN model checker (where '$\Box$' means "at all future moments"):

1. **Sense and Avoid**

$$\Box(\text{objectIsApproaching} \implies \{\text{direction} = \text{Right}\})$$

2. **Navigation in Aerodrome Airspace**

$$\Box\left[\begin{pmatrix} \text{changeHeading} \land \neg\text{objectIsApproaching} \\ \land\,\text{nearAerodrome} \land \neg\text{toldOtherwise} \end{pmatrix} \implies \neg\{\text{direction} = \text{Right}\}\right]$$

3. **ATC Clearance**

$$\Box(\{\text{state} = \text{TaxiingToRunwayHoldPosition}\} \implies \text{haveATCTaxiClearance})$$

3 Probabilistic Models of UAS Behaviour

Up to this point we have focused our verification on straightforward properties that the model does or does not satisfy. In real life examples of UAS operation, the situation is more complex: sensors may have errors; the UAS may not communicate with other on-board systems or other aircraft reliably; etc. We can begin to capture these notions of reliability in a formal way using a probabilistic model checker (e.g., PRISM [16]) and so gain a statistical measure of the level of compliance of a UAS control systems with the Rules of the Air. This is important because one element of the regulator-specified aircraft safety requirements is an inverse relationship between the probability of some failure occurring and the severity of its consequences.

3.1 Model Checking Rules of the Air in PRISM

In order to test the feasibility of probabilistic model checking for UAS, we created a model of the behaviour of a simple UAS using PRISM. Translation was relatively straightforward as PRISM shares a similar architecture to PROMELA for process modelling. However, PRISM is lower level and lacks some of the high-level features of PROMELA: standard output streams, message-passing channels, do–loops, etc., which presented a challenge for engineering and debugging the PRISM model.

Using PRISM we can assess temporal logic formulae similar to those used by SPIN, as well as statistical properties, and we were able to prove that our UAS model in PRISM satisfies the three Rules of the Air properties in the last section. Sample properties verified are as follows:

1. **Sense and Avoid**

$$\mathtt{P}^{>=1}\left[\ \mathtt{G}(\text{objectIsApproaching} \quad\Longrightarrow\quad \{\text{direction} = \text{Right}\})\ \right]$$

2. **Navigation in Aerodrome Airspace**

$$\mathtt{P}^{>=1}\left[\ \mathtt{G}\left(\begin{array}{c}\left(\begin{array}{c}\{\text{state} = \text{ChangingHeading}\} \wedge \neg\text{toldOtherwise} \\ \wedge\,\text{nearAerodrome} \wedge \neg\text{objectIsApproaching}\end{array}\right) \\ \Longrightarrow \\ \neg\{\text{direction} = \text{Right}\}\end{array}\right)\ \right]$$

3. **ATC Clearance**

$$\mathtt{P}^{>=1}\left[\ \mathtt{G}(\{\text{state} = \text{TaxiingToRunwayHoldPosition}\} \quad\Longrightarrow\quad \text{ATCTaxiClearanceGiven})\ \right]$$

Note that we use the PRISM operator $\mathtt{P}^{>=1}$, which tests that the subsequent property holds in all runs of the model, and $\mathtt{G}$ which means “globally” and is equivalent in meaning to the temporal “always in the future” operator.

3.2 Statistical Model Checking in PRISM

We again look at the rule concerning turning right when an object is approaching the UAS head-on. So far we have, essentially, an ideal sense-and-avoid system which is able to detect objects approaching head-on in a completely accurate way, i.e., with 100% certainty. However, real-life sense-and-avoid systems are unlikely to have this degree of accuracy so we define a sensor with *n*% accuracy as one which correctly identifies an approaching head-on object (requiring evasive action) *n*% of the time[1].

As we have been using ideal sensors thus far, we have been able to show that in all cases the UAS turns right when an object is approaching head-on. However, let us model a sensor with 95% accuracy. We do this as follows:

```
[MSGobjectApproaching] state=NormalFlight ->
     0.95: (state'=EmergencyAvoid) &
           (stateCache'=state) & (direction'=Right)
   + 0.05: (state'=NormalFlight);
```

This is the part of the UAS control system concerning what happens when the sensor unit sends an objectApproaching message (`MSGobjectApproaching`). We can then use the probabilistic model checking capabilities of PRISM to see how often this results in the UAS *not* going right when it should (i.e., when an object is approaching). We run the following query:

$$\mathtt{P}^{=?}\left[\ \mathtt{G}(\text{objectIsApproaching} \implies \{\text{state} = \text{EmergencyAvoid}\})\ \right]$$

This translates as, "What is the probability of the following property being satisfied in all states of the model: if an object is approaching head-on, then the state of the UAS is EmergencyAvoid?"[2] When we run this query PRISM returns the value 0.867, or 87%, indicating that a sensor accuracy of 95% causes the UAS to turn right on only 87% of the paths through the simulation where an object is approaching head-on.

3.2.1 Statistical Model Checking and Certification

Results of the kind shown above can be very useful from a certification perspective, as we can predict the effect of the failure of some part of the UAS, e.g., a sensor, in a quantitative way. This is relevant as certification trials for aircraft often require a quantifiable level of certainty, for example a single point failure on an aircraft must not manifest itself more often than every 10^9 flying hours if its failure could cause a catastrophic failure condition for the aircraft.

[1]We are only modelling false negatives, not false positives, as the latter would be less problematic regarding this Rule of the Air.

[2]The `EmergencyAvoid` state always results in the UAS turning right.

4 *Reasoning* Agent Models of UAS Behaviour

The reactive UAS agent models presented so far, written in PROMELA and PRISM, are simple in terms of autonomy: in both cases the UAS follows a series of reflexive responses to environmental changes, e.g., a message has come from ATC saying taxi clearance has been given, so update the UAS state to "Taxiing." It may be desirable to encode more complex autonomous behaviours based on ideas from intelligent agent theory, such as the Beliefs–Desires–Intentions (BDI) framework for autonomous agents. Such approaches offer a natural way of specifying, engineering and debugging autonomous behaviour [23]. Another advantage is model checking autonomous behaviour: we can see the state of the agent's beliefs, desires and intentions at the point a property was violated.

Gwendolen is a BDI agent programming language developed by researchers at the Universities of Liverpool and Durham and is designed specifically for agent verification [10]. Gwendolen agents consist of beliefs, goals, intentions and plans. (Goals are desires which are being actively pursued.) Each plan consists of a triggering event, a guard and a number of deeds which are executed if the plan is triggered and the guard is satisfied. A Gwendolen agent begins with sets of initial beliefs and goals, and a set of plans. The agent selects a subset of plans based on its beliefs about the current situation and its current goals, i.e., what it wants to achieve. For instance, the following is a very basic Gwendolen agent model of a UAS control system:

AGENT: `EXEC`
Initial Beliefs: $waitingAtRamp, my\_name(uav)$
Initial Goals: $+!_p requestTaxiClearance$
Plans:

$+!_p requestTaxiClearance : \mathsf{B}\ my\_name(Name) \wedge \neg\uparrow^{atc} \mathbf{tell}(Name, requestTaxi)$
$\quad\quad$ <- $\uparrow^{atc} \mathbf{tell}(Name, requestTaxi)$

This simple UAS agent believes initially that it is waiting at the ramp at the beginning of its mission. It has an initial goal — to request taxi clearance — and a single plan: if a goal to request taxi clearance is added (the trigger), it plans to send a message to the ATC requesting taxi clearance (the deed) as long as it hasn't already sent such a message (the guard).

Along these lines we have constructed a model of a UAS agent written in Gwendolen. Our UAS agent is much more complex, consisting of 36 different plans as opposed to the one-plan agent above. The UAS is similar in behaviour to the agents written in PROMELA and PRISM: it taxis, holds, lines up and takes off, and once airborne it performs simple navigation and sense/avoid actions. Finally, it lands. The chief difference is that these behaviours are specified in terms of beliefs, desires and intentions, which provide a richer language for describing autonomous behaviour. For instance, "the UAS is taxiing", "the UAS wants to taxi", "the UAS believes it is taxiing", and "the UAS intends to taxi", are all distinct for a BDI agent. Furthermore it is potentially possible to reason about other agents' beliefs, such as "the UAS believes that the ATC believes the UAS is taxiing", allowing for richer interactions between different parts of the model than is found with similar processes in PROMELA or PRISM.

The trade-off is that whilst BDI agent software is more representative of natural-world intelligent systems and provides an improved facility for describing autonomous systems, the added complexity of the agent programs makes model checking much slower. In general, we talk in terms of minutes and hours for verifying UAS agent programs, as opposed to milliseconds for the simpler PROMELA and PRISM models.

4.1 Model Checking Reasoning UAS Agents

Agents are often written in agent programming languages, so we need an agent model checker to verify agent programs [4]. We use AJPF (for Agent JPF), which works by providing a Java interface for BDI agent programming languages called the Agent Infrastructure Layer (AIL) [17]. Interpreters for agent programming languages are written using the AIL, and the resulting Java program can then be verified via AJPF [2]. AJPF is, in turn, built on JPF, the Java PathFinder model checker developed at NASA Ames Research Center [22, 15]. For example, an agent program written in Gwendolen is executed by an interpreter written in Java and using the AIL. Properties can then be checked against the model using AJPF.

We verified our UAS agent model using this method. For consistency we used the same subset of the Rules of the Air used for the PROMELA and PRISM UAS models. The results are summarised as follows.

1. **Sense and Avoid**

$$\Box(\mathsf{B}(\text{exec}, \text{objectIsApproaching}) \implies \mathsf{B}(\text{exec}, \text{direction}(\text{right})))$$

2. **Navigation in Aerodrome Airspace**

$$\Box(\mathsf{B}(\text{exec}, \text{changeHeading}) \land \mathsf{B}(\text{exec}, \text{nearAerodrome}) \land \neg\mathsf{B}(\text{exec}, \text{toldOtherwise}) \\ \implies \neg\mathsf{B}(\text{exec}, \text{direction}(\text{right})))$$

3. **ATC Clearance**

$$\Box(\mathsf{B}(\text{exec}, \text{taxiing}) \implies \neg\mathsf{B}(\text{exec}, \text{taxiClearanceGiven}))$$

Here we use the belief operator B to specify beliefs about the agents being verified. This property translates as, “It is always the case that if the agent ‘exec’ believes that an object is approaching, then it also believes that its direction is right.”

4.1.1 Detecting Errors in UAS Implementation

In order to test the usefulness of our UAS model, we introduced a minor error into the code to simulate a typical software engineering error. Normally, when the

UAS has discovered that there is an object approaching head-on and that it should also change heading it prioritises the former, as avoiding a potential crash takes precedence over navigation. However, our error caused the UAS to have no such priority. The net effect on the UAS behaviour is that it would start to turn right to avoid the object, but would then turn left to navigate (as it was within aerodrome airspace). Specifically, the errant code was as follows:

$$\begin{aligned}&+!_p makeDecision(objectApproaching, changeHeading) : \mathsf{B}\ normalFlight(X)\\ &\qquad <-\ +!_p handleObjAppr(X), +!_p handleChangeHeading(X)\end{aligned}$$

The model checker discovered the fault when we verified the Sense-and-Avoid property (see above).

4.2 Model Checking Advanced Autonomy in UAS Agents

The UAS agent model constructed so far will always turn right when an object is approaching head-on. This is in accordance with the Rules of the Air. However there may be occasions when it is advantageous (or indeed necessary) for the UAS agent to disobey certain Rules of the Air in order to maintain a safe situation. For instance, consider the case where an object is approaching head-on, and the UAS agent “knows” it should turn to the right. However, the approaching aircraft may indicate that its intention is to turn to the left (e.g., by initiating a roll to the left, manifested by its left wing dropping). At this point a rational pilot would assume that the other aircraft is going to turn left, and would realise that turning right would greatly increase the possibility of a collision. Turning left would be the more rational action to take. Likewise, if the other aircraft’s intention is to turn right, the rational action is to turn right. If the intention is unknown, then the rational action is to follow the Rules of the Air, i.e., turn right.

We added several plans to our UAS agent model in order to make the agent adopt this advanced autonomous behaviour. The sensor unit was re-written, so that instead of sending an “object approaching head-on” message, it now sends information about intentions, e.g., “object approaching head-on and its intention is to go left.” The UAS was then enhanced to take into account beliefs about the other object’s intention when making a decision about which way to go when an object is approaching head-on:

$$\begin{aligned}&+!_p makeDecision(objectApproaching(N, intentionTurnLeft), changeHeading) :\\ &\quad \mathsf{B}\ normalFlight(X)\ <-\ +intention(N, turnLeft), +!_p handleObjAppr(X)\end{aligned}$$

In other words, “When the Executive has to decide between an object approaching head on (and intending to turn left) and a directive from the navigation manager to change heading, and the Executive believes it is in normal flight mode, it will add the belief that the object’s intention is to turn left, and will add as a goal to handle the object approaching by taking evasive action.”

Therefore, adding advanced autonomy will cause the UAS agent to disobey the Rule of the Air concerning turning right when an object is approaching head-on in the name of safety. The reason is that there will be times when there is an object approaching head on, but the UAS turns left because it has detected the intention of the object is to turn left. For this reason we must modify the properties being checked. For instance the rule in Section 4.1 concerning turning right when there is an object approaching head-on becomes:

$$\square(\mathsf{B}(\text{uav}, \text{objectIsApproaching}) \land \mathsf{B}(\text{uav}, \text{intention}(\text{right})) \implies \mathsf{B}(\text{uav}, \text{direction}(\text{right})))$$

In other words, "It is always the case that if the UAS believes there is an object approaching head on and the intention of the object is to turn right, then the UAS turns right." We tested similar properties for the cases where the intention is to turn left and where the intention is unknown, finding that the agent satisfied all three cases, as well as the "Navigation in Aerodrome Airspace" and "ATC Clearance" properties.

It is important to note that in practice there is no conflict between this advanced autonomous behaviour and the Rules of the Air, as the advanced behaviour is similar to what would be expected of a human pilot. All Rules of the Air are subject to interpretation, i.e., the previously mentioned *airmanship*; there are times when the strict Rules of the Air must be disobeyed to maintain safe operations.

5 Conclusion

We have constructed basic agent models of Unmanned Aircraft Systems for three different model checking platforms: PROMELA / SPIN for symbolic model checking, PRISM for probabilistic symbolic model checking and Gwendolen / AJPF for agent model checking. In each case we tested our UAS model against a small subset of the Rules of the Air corresponding to the following cases: (i) Sense and Avoid; (ii) Navigation in Aerodrome Airspace; and (iii) ATC Clearance. These rules were chosen because they present interesting cases of UAS autonomy: sense-and-avoid and "human in the loop" cases (rules (i) and (iii) respectively) are essential for UAS engineering [19]. In addition, rules (i) and (ii) are interesting because they are potentially conflicting, presenting an interesting challenge for engineering and verification.

The models we constructed in SPIN / PROMELA and PRISM were very fast in terms of verification, requiring only milliseconds and megabytes to model-check a Rule of the Air. However, their low-level process-modelling and state-transition systems presented problems when it came to modelling more advanced autonomy, as this is something for which those verification systems were not designed. Agent languages in the BDI tradition (Gwendolen being one such example) allow faster and more accurate engineering of autonomous systems, but this comes at a price: in our example, the time required for verification of a Rule of the Air increased to minutes and hours.

The models we have used are very simple and naive, and the temporal requirements are very straightforward. However, since most of the elements within the UAS control system are likely to be similarly simple and since quite a number of Rules of the Air are similarly straightforward, then our preliminary results suggest that it is indeed feasible to use formal methods (and model checking in particular) to establish UAS compliance with the at least *some* of the Rules of the Air. The areas where the models/designs might be more sophisticated and where the Rules of the Air go beyond a straightforward representation are considered in the subsequent sections of future work. We are confident that undertaking this further work will indeed move us much closer to acceptable certification for autonomous UAS.

A possible disadvantage of our approach, from the perspective of certification of airworthiness, is that for an existing UAS agent (written in some compiled language such as SPARK Ada) any models written in PROMELA, PRISM and Gwendolen may not be accurate, so that the verification process will not lead to useful evidence for certification. One possible way to avoid this problem is to specify the agent architecture using a process modelling language, and then use a formal software development methodology to accurately implement the specification. Alternatively, in the case of AJPF, implementation may not even be necessary as the result of the verification process is code executable within a Java virtual machine — the agent is effectively already implemented.

5.1 Related Work

There have been a number of applications of formal methods to UAS. For example, Sward used SPARK Ada to prove correctness of UAV cooperative software [21]; Chaudemar et al. use the Event-B formalism to describe safety architectures for autonomous UAVs [6]; Jeyaraman et al. use Kripke models to model the activity of multi-UAV teams and use the PROMELA model checker to verify safety and reachability properties amongst others [13]; Sirigineedi et al. use Kripke models to model UAV cooperative search missions, and used the SMV model checker to prove that the UAVs do not violate key safety properties [20].

Formal methods have also been applied to autonomous systems in the aerospace domain, e.g., Pike et al. describe an approach to V&V of UAVs using lightweight domain-specific languages; Brat et al. use the PolySpace C++ Verifier and the assume–guarantee framework to verify autonomous systems for space applications [5]; while Bordini et al. proposed the use of model checkers to verify human–robot teamwork in space [3].

5.2 Future Work

In this paper we have modelled a UAS with only basic functionality. Adding functionality would presumably add complexity to the model and increase verification time, although quantifying this is difficult without having a more complete model to hand. Finally, for a complete test of UAS airworthiness we also need to verify the UAS subsystems with which our "Executive" communicates: various avionics systems including sensors, actuators and automatic flight control systems would all need to be certified separately and together, presumably using existing methods such as SPARK Ada, for example.

However, an obvious next step is to expand the functionality of the UAS as we have described it, and test whether it is possible to verify it against a larger subset of the Rules of the Air. Another interesting avenue would be to obtain "real-life" UAS source code, or an abstract state transition system describing the behaviour of an already-operational UAS, and generate a model of its control system in order to verify different aspects of its airworthiness.

An immediate aim of the Virtual Engineering Centre is to create a virtual prototype of a complete autonomous UAS, including including agent, UAV, complex flight control system, sensors, avionics, ATC and cockpit display, and take it through the development and certification phases of the virtual engineering life cycle. The resulting prototype UAS can then be used as a test-bed for demonstrating a range of systems critical to autonomous flight, e.g., novel sensor systems.

Acknowledgements

This work is supported through the Virtual Engineering Centre (VEC), which is a University of Liverpool initiative in partnership with the Northwest Aerospace Alliance, the Science and Technology Facilities Council (Daresbury Laboratory), BAE Systems, Morson Projects and Airbus (UK). The VEC is funded by the Northwest Regional Development Agency (NWDA) and European Regional Development Fund (ERDF) to provide a focal point for virtual engineering research, education and skills development, best practice demonstration, and knowledge transfer to the aerospace sector.

The authors would like to thank Charles Patchett (Group Leader, Autonomous Systems Research Group) and Ben Gorry (Head of Capability, Software and Safety Technology Group) of BAE Systems for their guidance and support.

References

[1] R. Bordini, M. Dastani, J. Dix, and A. El Fallah-Seghrouchni, editors. *Multi-Agent Programming: Languages, Tools and Applications*. Springer, 2009.

[2] R. H. Bordini, L. A. Dennis, B. Farwer, and M. Fisher. Automated Verification of Multi-Agent Programs. In *Proc. 23rd Int. Conf. Automated Software Engineering (ASE)*, pages 69–78. IEEE Computer Society Press, 2008.

[3] R. H. Bordini, M. Fisher, and M. Sierhuis. Formal Verification of Human-Robot Teamwork. In *Proc. 4th Int. Conf. Human-Robot Interaction (HRI)*, pages 267–268. ACM, 2009.

[4] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Model Checking Rational Agents. *IEEE Intelligent Systems*, 19(5):46–52, 2004.

[5] G. Brat, E. Denney, D. Giannakopoulou, J. Frank, and A. Jonsson. Verification of Autonomous Systems for Space Applications. In *Proc. IEEE Aerospace Conference*, 2006.

[6] J.-C. Chaudemar, E. Bensana, and C. Seguin. Model Based Safety Analysis for an Unmanned Aerial System. In *Proc. Dependable Robots in Human Environments (DRHE)*, 2010.

[7] Civil Aviation Authority. CAP 393 Air Navigation: The Order and the Regulations. `http://www.caa.co.uk/docs/33/CAP393.pdf`, April 2010.

[8] Civil Aviation Authority. CAP 722 Unmanned Aircraft System Operations in UK Airspace — Guidance. `http://www.caa.co.uk/docs/33/CAP722.pdf`, April 2010.

[9] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[10] L. A. Dennis and B. Farwer. Gwendolen: A BDI Language for Verifiable Agents. In *Logic and the Simulation of Interaction and Reasoning*. AISB'08 Workshop, 2008.

[11] European Aviation Safety Agency. Certification Specifications for Large Aeroplanes CS-25, October 2003. ED Decision 2003/2/RM Final 17/10/2003.

[12] G. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison–Wesley, 2004.

[13] S. Jeyaraman, A. Tsourdos, R. Zbikowski, and B. White. Formal Techniques for the Modelling and Validation of a Co-operating UAV Team that uses Dubins Set for Path Planning. In *Proc. American Control Conference*, 2005.

[14] C. Johnson. Computational Concerns in the Integration of Unmanned Airborne Systems into Controlled Airspace. In *Proc. 29th Int. Conf. Computer Safety, Reliability and Security (SAFECOMP)*, volume 6351 of *LNCS*. Springer, 2010.

[15] Java PathFinder. `http://javapathfinder.sourceforge.net`.

[16] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic Symbolic Model Checker. In *Proc. 12th Int. Conf. Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS)*, volume 2324 of *LNCS*, pages 200–204. Springer, 2002.

[17] Model-Checking Agent Programming Languages. `http://mcapl.sourceforge.net`.

[18] Office of the Secretary of Defense. Unmanned Aircraft Systems Roadmap 2005–2030. US DoD Publication, 2005.

[19] C. Patchett and D. Ansell. The Development of an Advanced Autonomous Integrated Mission System for Uninhabited Air Systems to Meet UK Airspace Requirements. In *Proc. International Conference on Intelligent Systems, Modelling and Simulation*, 2010.

[20] G. Sirigineedi, A. Tsourdos, R. Zbikowski, and B. A. White. Modelling and Verification of Multiple UAV Mission Using SMV. In *Proc. FM-09 Workshop on Formal Methods for Aerospace*, volume 20 of *EPTCS*, 2009.

[21] R. E. Sward. Proving Correctness of Unmanned Aerial Vehicle Cooperative Software. In *Proc. IEEE International Conference on Networking, Sensing and Control*, 2005.

[22] W. Visser, K. Havelund, G. P. Brat, S. Park, and F. Lerda. Model Checking Programs. *Automated Software Engineering*, 10(2):203–232, 2003.

[23] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2002.