Formal Methods for the Certification of Autonomous Unmanned Aircraft Systems

Matt Webster[1,4], 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
[4] Corresponding author. Email: matt@liv.ac.uk, Tel/Fax: +44 (0) 1925 864850

Abstract. In this paper we assess the feasibility of using formal methods, and model checking in particular, for the certification of Unmanned Aircraft Systems (UAS) within 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. Next we build a more advanced 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.

Keywords: Model Checking, Formal Methods, Unmanned Aircraft System, Autonomous Systems, Certification

1 Introduction

An Unmanned Aircraft System (UAS, plural UAS) is a group of 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. UAS are now routinely used in military applications, their key advantages coming from their ability to be used in the so-called "dull, dangerous and dirty" missions, e.g., long duration/persistence flights and flights into hostile or hazardous areas (such as clouds of radioactive material) [20]. There is a growing acceptance, however, 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. [21,15]. However, in order for this integration to take place in a meaningful way, UAS must be capable of routinely flying through "non-segregated" airspace. Today, for most useful civil applications, UAS can fly in UK civil airspace but in what is known as segregated airspace, that is, airspace which is for the exclusive use of the specific user. For routine UAS operations, this will not be 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 meeting this regulatory challenge, 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 indeed 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, [12]). 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 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, we investigate the potential usefulness of model checking in providing formal evidence for the certification of UAS. The work described here develops a new approach and describes 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 (in accordance with the Rules of the Air), could provide powerful evidence to a regulator that the UAS would not compromise the safety of other airspace users. Thus, the work described herein may be a first step in answering the question as to whether or not formal verification tools have the potential to contribute to this overall ambition.

This is but one step towards the certification of autonomous UAS in non-segregated UK airspace, yet it allows us to show how a route to full certification might be relieved of some of the burden of analysis/testing required at present. This can save time and increase reliability, but might come at the cost of an increased level of expertise required of the analysts involved in the certification process. In particular, we focus on using formal methods to verify the high-level "decision-making" aspects of autonomous UAS control which may eventually complement or replace human decision-making for UAS. The model checking approaches we describe could help to establish the robustness of a given decision-making system, and when combined with existing approaches to aircraft software engineering, could provide a route to certification of autonomous UAS.

1.1 Approach

Since the route to airframe and automatic flight 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 would be to show that this core autonomous control (in the form of an "intelligent" agent) would make the same decisions as a human pilot/controller *should* 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 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. It should be recognised that demonstrating that an autonomous agent's

decisions will conform to the Rules of the Air is not the same as providing sufficient evidence for certification. However, demonstrating that this is the case will provide one piece of evidence that will support any application for certification of a system.

To show how this might be done, we chose a small subset of the Rules of the Air and encoded these in a formal logic. ("The Rules of the Air Regulations 2007," is large: around 15,000 words plus accompanying images [7].) We modelled a UAS control system as an *executable agent model* (initially using PROMELA [13], but later in a higher-level agent language [1]), and applied model checking to verify that the UAS agent satisfied the selected subset of the Rules of the Air.

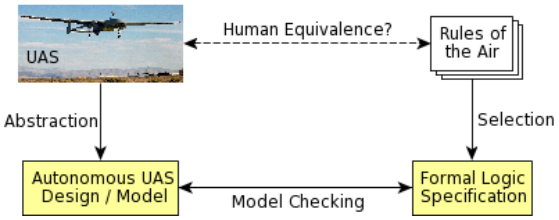

Fig. 1. An approach to certification via the Rules of the Air. (Image: SSgt. R. Ramon, USAF.)

Our approach is summarised in Fig. 1. 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 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 verification and validation (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* of 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 within this study. Finally, some of the rules implicitly use quite complex notions, such as "likelihood", "knowledge", "the other pilot's intention", "expectation", and so on (see below for some examples). 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.

1.2 Paper Structure

In Section 2 we describe the software tools to be used for UAS agent verification and describe how the small subset of the Rules of the Air for verification was chosen. Then, in Section 3 we model a basic UAS agent in PROMELA, and verify it against a small subset of the Rules of the Air using the SPIN model checker. 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 [27]. 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, initially, we simply equate "agent" with "process". Thus, we model the UAS's choices/decisions as a single process in PROMELA, and use SPIN to show that the UAS agent satisfies the selected subset of the Rules of the Air.

In Section 4 we construct a UAS control system based on a rational agent model. This is written using the autonomous agent language Gwendolen [10], and we show that it can be verified against the same Rules of the Air using the agent model checker AJPF [2,11]. We introduce more advanced autonomous behaviour into the UAS agent, and verify that this acts in accordance with the subset of the Rules of the Air.

There are two main reasons for using a rational agent model. 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 UAS's situation but also the agent's beliefs about the intentions of other UAS/aircraft. The second reason is 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, in Section 4 we show how we might "tease" out such aspects into formal specifications involving intentions/beliefs that could then be checked through our verification system.

Finally, in Section 5 we compare the different approaches to UAS agent modelling and verification, and we present directions for future research.

2 Model Checking

Model checking is a variety of formal verification in which a logical property is exhaustively evaluated against all executions of a system [9]. Typically, the logical property is expressed within a *temporal* logic. This allows us to refer to properties that occur now, in the next moment, and at selected moments in the future. As well as classical logic operators, temporal logic also provides operators such as '$\Box$', meaning "at all future moments". Thus, "$\Box(x \Rightarrow y)$" means that at all future moments within the execution, if x is true then y must be true. This is distinct from "$x \Rightarrow \Box y$" which means that, if x is true then y must be true at all future moments.

In the model checker we first utilise, called SPIN [13], the program to be checked is written in the PROMELA programming language. The SPIN model checker then exhaustively checks our required temporal formula against all possible executions of the program. If successful, this means that no matter how the program executes, the required property will still be true. However, if the model checker finds a specific execution that violates the required property, it identifies this to the user.

Although we begin by using the PROMELA language and SPIN for verification, we later use a more sophisticated language, Gwendolen [10], a high-level agent-based programming language, to develop more advanced UAS control. We check the Gwendolen program against the same logical requirements, but as SPIN only checks PROMELA programs, we must use a different model checker called AJPF [2,11] to establish correctness of the Gwendolen program with respect to the logical properties.

2.1 Selecting Rules of the Air for Model Checking

We chose a small subset of just three Rules of the Air [7] 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 might potentially come into 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 describe 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 [21]. 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 (1 or 2) it chooses to obey.

Simplification was necessary to encode the above "rules" so that they could be model checked. For instance, in the second rule, there are a number of factors which could "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", and not to model these factors separately.

3 Reactive UAS Agents

Through consultations with researchers from the Autonomous Systems Research Group at BAE Systems (Warton, UK) 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 "Sense and Avoid" scenario described in Section 2.1. Clearly there would 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 the ATC, it will set its flight phase to "taxi" and start taxiing to the runway where it will wait for 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 controlling a UAS.)

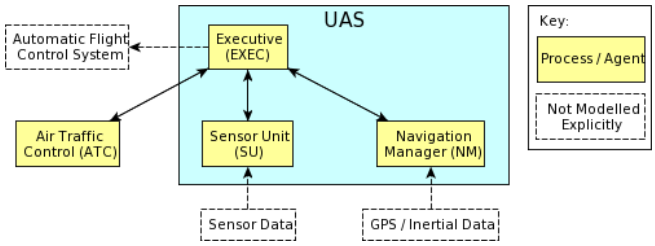

Fig. 2. UAS Models in PROMELA and Gwendolen. Arrows represent information flow.

3.1 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 [13]. The UAS is divided into a number of components: the Executive, the Sensor Unit (SU) and the Navigation Manager (NM). In Fig. 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 course. 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 Fig. 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 has different values to represent the different parts of the UAS's mission: `WaitingAtRamp` (start of mission), `TaxiingToRunwayHold-Position`, `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 changes 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 advising 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.

3.2 Model Checking the Rules of the Air in SPIN

As we have a system capturing selected behaviour within 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 2.1 using the SPIN model checker. The temporal logic form of these three rules are as follows.

1. **Sense and Avoid:** $\Box(\text{objectIsApproaching} \implies \{\text{direction} = \text{Right}\})$
2. **Navigation in Aerodrome Airspace:**

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

3. **ATC Clearance:** $\Box(\{\text{state} = \text{TaxiingToRunwayHoldPosition}\} \implies \text{haveATCTaxiClearance})$

The UAS agent model was found to satisfy all three properties.

4 Rational UAS Agents

The reactive UAS agent model presented so far, written in PROMELA, is quite basic in terms of autonomy. 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 [22]. As suggested by the name, agents comprise *beliefs* (i.e., their information about the world), *desires* (i.e., their long term aims), and *intentions* (i.e., the things the agent is doing to try to achieve its desires). Such approaches offer a natural way of specifying, engineering and debugging high-level autonomous behaviour [27]. Another advantage is model checking

autonomous behaviour: we can see the state of the agent's beliefs, desires and intentions at the point a particular logical property is violated.

To model BDI agents we use a BDI agent language as PROMELA is not designed for this purpose. To use PROMELA in this way, beliefs, desires and intentions would have to be constructed from the native PROMELA constructs such as processes, variables, etc. In contrast, BDI agent languages have these features "built-in". Therefore the software engineer is more able to focus on the behaviours of the autonomous system when using a BDI agent language than when using PROMELA. Likewise, the SPIN model checker used to verify PROMELA programs does not contain any operators concerning agents' beliefs, desires or intentions, whereas agent model checkers let us specify different agents' beliefs, desires and intentions within the property being checked.

Gwendolen [10] is a BDI agent programming language designed specifically for agent verification. 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.

We have constructed a model of a UAS agent written in Gwendolen. Our UAS agent consists of 44 different plans, several of which are shown below. The UAS is similar in behaviour to the agents written in PROMELA: it taxies, holds, lines up and takes off, and once airborne it performs simple navigation and sense/avoid actions. Finally, it lands. The UAS agent believes initially that it is waiting at the ramp at the beginning of its mission, and that it has no forward direction. It has an initial goal (here, "$!_p$" means a goal to perform some action) — to run the "startup procedure" — and a set of plans. For instance, the first plan says that if a belief that normal flight is underway is added (the trigger, $+normalFlight$), the agent will delete the last message from the sensor unit ($-su(S)$) and will undertake an action to send a message to the sensor unit requesting information ($send(su, poll)$). The "$\{\ldots\}$" in this case is a guard condition on the plan. Here '$\top$' always evaluates to "True".

Agent: *exec*
Initial Beliefs: *waitingAtRamp*, *direction*(*none*)
Initial Goals: $!_p startup$
Plans:
$+normalFlight$: $\{\top\} \leftarrow -su(S),\ send(su, poll)$;
$+!_p\ pollAgents$: $\{\top\} \leftarrow -su(S),\ send(su, poll)$;
$+su(S)$: $\{\top\} \leftarrow -nm(N), send(nm,poll)$;
$+nm(N)$: $\{B\ su(S),\ \neg G\ makeDecision(S,N)\ \} \leftarrow +!_p\ makeDecision(S,N)$;
$+nm(X,N)$: $\{B\ su(X,S),\ \neg G\ makeDecision(S,N)\ \} \leftarrow +!_p\ makeDecision(S,N)$;
$+!_p\ makeDecision(objAppr,headingOk) : \{B\ normalFlight\} \leftarrow +!_p\ handleObjAppr$;
$+!_p\ makeDecision(objAppr,changeHeading) : \{B\ normalFlight\} \leftarrow +!_p\ handleObjAppr$;
$+!_p\ handleObjAppr : \{B\ normalFlight,\ B\ direction\ (D)\} \leftarrow -normalFlight,$ lock,
$-direction\,(D),\ +direction\,(\ right\ ),$ unlock, $+emergencyAvoid, +objectIsApproaching,$
$+!_p\ pollAgents$;

The chief difference between the Gwendolen and PROMELA models is that the Executive's 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 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.

The trade-off is that whilst BDI agent software is more representative of natural-world intelligent systems and provides improved expressiveness for describing autonomous systems, the added complexity of the agent programs makes subsequent 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 programs.

In our implementation the architecture of the Gwendolen UAS model is slightly different from the PROMELA model. Firstly, we modelled the Executive as a Gwendolen agent, but the ATC, NM and SU were modelled within the agent's Java environment. The reason for this was that it makes intuitive sense; the Executive is the autonomous part of the model on which we focus our model checking efforts, and therefore is programmed in Gwendolen, a language for autonomous agents. Also, a future objective is to be able to connect the Executive to simulated sensors and navigation systems within a networked simulation environment, and replacing the Java simulated environment with a networked simulated environment was simpler than connecting Gwendolen agent models of the SU, NM and ATC to the networked simulation environment. In model checking terms there is little difference; the simulated SU, NM and ATC in Java perform the same function as the corresponding processes in PROMELA, and enable the full state space of the Executive to be explored.

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 [2,11]. AJPF is, in turn, built on JPF, the Java PathFinder model checker developed at NASA Ames Research Center [25,16]. For example, an agent program written in Gwendolen is executed by an interpreter written in Java and using the AIL. Temporal 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 earlier used for the PROMELA UAS model. The properties verified are 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:**
   $\Box(\mathsf{B}(\text{exec}, \text{changeHeading}) \wedge \mathsf{B}(\text{exec}, \text{nearAerodrome}) \wedge \neg\mathsf{B}(\text{exec}, \text{toldOtherwise}) \implies \neg\mathsf{B}(\text{exec}, \text{direction}(\text{right})))$
3. **ATC Clearance:** $\Box(\mathsf{B}(\text{exec}, \text{taxiing}) \implies \mathsf{B}(\text{exec}, \text{taxiClearanceGiven}))$

Here we use the belief operator 'B' to specify beliefs about the agents being verified, e.g., property 1 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 to the right."

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 collision 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:

$+!_p$ *makeDecision*(*objAppr*,*changeHeading*){ $B\neg$*normalFlight*(X) } $\leftarrow$ $+!_p$ *handleObjAppr*(X),
$+!_p$ *handleChangeHeading*(X);

The model checker found the fault when we verified the "Sense and Avoid" property.

4.2 Model Checking More 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 more 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 intentions when making a decision about which way to go when an object is approaching head-on:

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

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." Adding such 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:

$$\Box(\mathsf{B}(\text{exec}, \text{objectIsApproaching}) \wedge \mathsf{B}(\text{exec}, \text{intention}(\text{right})) \Longrightarrow \mathsf{B}(\text{exec}, \text{direction}(\text{right})))$$

In other words, "It is always the case that if the Executive believes there is an object approaching head on and the intention of the object is to turn right, then the UAS turns right." We verified 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 in order to maintain safe operations.

5 Conclusions

We have constructed basic agent models of Unmanned Aircraft Systems for two different model checking platforms: PROMELA / SPIN for standard 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:

1. Sense and Avoid;
2. Navigation in Aerodrome Airspace; and
3. Air Traffic Control Clearance.

These rules were chosen as interesting cases of UAS autonomy: "Sense and Avoid" and "human in the loop" cases (rules 1 and 3 respectively) are essential for UAS engineering [21]. In addition, rules 1 and 2 are interesting because they are potentially conflicting, presenting an interesting challenge for engineering and verification.

The model we constructed in SPIN / PROMELA was very fast in terms of verification, requiring only milliseconds and megabytes to model-check a Rule of the Air. However, its 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 single Rule of the Air property increased to minutes and hours.

The models and temporal requirements we have used are relatively 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 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 section of future work. We are confident that this approach can move us towards 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 a compiled language such as SPARK Ada) any models written in PROMELA or Gwendolen may not be accurate,

so that the verification process will not lead to useful evidence for certification. A well-known 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.

Another possible difficulty is in justifying the abstractions made during the modelling process. Applying our approach to a given autonomous UAS control system requires modelling the system, e.g., using PROMELA or the Gwendolen agent language. The conclusions drawn from model checking are only as useful as our confidence in the model itself; therefore model validation is important when applying our approach to implemented autonomous UAS systems. For similar reasons, the properties used for model checking would need to be validated with respect to required standards of behaviour.

5.1 Impact

Two principal questions for UAS manufacturers are whether Formal Methods has anything to offer autonomous UAS, and if so, what kind of approaches should be used and in what manner? These are the questions that we have started to answer but the answer is by no means complete; the construction of the models described in the paper has shown that the SPIN and Agent JPF model checkers are well-suited to the task of specifying and analysing autonomous UAS behaviour. Furthermore, the paper demonstrates that these models can be checked to be in accordance with a small subset of the Rules of the Air, a statutory document specifying many of the requirements of pilots and aircraft in UK airspace. Therefore the paper has demonstrated that Formal Methods could indeed be useful for providing evidence to regulatory authorities that a given autonomous UAS is airworthy and presents no additional risks beyond those currently encountered by traditional manned aircraft. This is a small but crucial first step on the road to certification, which is likely to require intensive investigation by both academic and industrial researchers over the coming years. This work has begun to show how the problem of verifying that an autonomous computer system is equivalent to a human might be tackled.

5.2 Related and Future Work

This paper has focused on the problem of engineering and certification of autonomous UAS, with the emphasis on verification of high-level decision making. However there is a wealth of literature in the field of control engineering concerning automatic flight control systems (e.g., autopilot, autoland) designed to assist the safe operation of manned vehicles [18]. In addition there is much in the literature concerning Airborne Collision Avoidance Systems (ACAS) which have tackled the sense-and-avoid problem, primarily in the arena of manned aircraft [26]. In this paper we attempted to formalise Rules of the Air (written in natural language) to derive properties describing the desired behaviour of autonomous UAS. These properties could then be checked against a model of an autonomous UAS control system. Deriving formal specifications from requirements written in natural language has also been examined elsewhere, e.g., [19].

There have been several uses of formal methods in UAS. For example: Sward used SPARK Ada to prove correctness of UAV cooperative software [24]; Chaudemar et al. use the Event-B formalism to describe safety architectures for autonomous UAVs [6]; Jeyaraman et al. use Kripke models to model multi-UAV teams and use SPIN to verify safety and reachability properties amongst others [14]; Sirigineedi et al. use Kripke models to model UAV cooperative search missions, and use the SMV model checker to show that the UAVs do not violate key safety properties [23]. Formal methods have also been applied to autonomous systems in the aerospace domain: 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]. Importantly, none of these use formal verification to establish that an autonomous systems is "equivalent" (even to a limited extent) to a human pilot, as we do here.

In this paper we have only modelled a very basic UAS. Adding functionality would add complexity to the model and likely increase verification time, although quantifying this is difficult without having a more complete model to hand. 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.

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 increasingly large subsets 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.

A key area for future research is in the management of complexity: as the complexity of the model of autonomous UAS behaviour increases, so will the time and space required for verification by the model checker. However it is possible that novel abstractions, modelling techniques and advances in computer technology and model checking software will mitigate this problem.

An immediate aim is to use the formally verified Executive agent within a virtual prototype of an autonomous UAS, including agent(s), UAV, complex flight control system, sensors and ground control station, and test whether Monte Carlo methods can be used to quantify UAS behaviour and provide evidence for certification.

Acknowledgements The authors would like to thank Charles Patchett and Ben Gorry of BAE Systems (Warton) for their guidance and support.

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.

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. L. A. Dennis, M. Fisher, M. P. Webster, and R. H. Bordini. Model Checking Agent Programming Languages. *Automated Software Engineering*. In press.
12. European Aviation Safety Agency. Certification Specifications for Large Aeroplanes CS-25, October 2003. ED Decision 2003/2/RM Final 17/10/2003.
13. G. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. AW, 2004.
14. 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.
15. 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.
16. Java PathFinder. http://javapathfinder.sourceforge.net.
17. Model-Checking Agent Programming Languages. http://mcapl.sourceforge.net.
18. D. McRuer and D. Graham. Flight control century: Triumphs of the systems approach. *Journal of Guidance, Control and Dynamics*, 27(2):161–173, 2004.
19. A. P. Nikora and G. Balcom. Automated identification of LTL patterns in natural language requirements. In *Proceedings of the 20th International Symposium on Software Reliability Engineering*, ISSRE '09, pages 185–194. IEEE Computer Society, 2009.
20. Office of the Secretary of Defense. Unmanned Aircraft Systems Roadmap 2005–2030. US DoD Publication, 2005.
21. 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.
22. A. Rao and M. Georgeff. Modeling Agents within a BDI-Architecture. In *Proc. 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 473–484. Morgan Kaufmann, 1991.
23. G. Sirigineedi, A. Tsourdos, R. Zbikowski, and B. A. White. Modelling and Verification of Multiple UAV Mission Using SMV. In *Proc. FMA-09*, volume 20 of *EPTCS*, 2009.
24. R. E. Sward. Proving Correctness of Unmanned Aerial Vehicle Cooperative Software. In *Proc. IEEE International Conference on Networking, Sensing and Control*, 2005.
25. W. Visser, K. Havelund, G. P. Brat, S. Park, and F. Lerda. Model Checking Programs. *Automated Software Engineering*, 10(2):203–232, 2003.
26. E. Williams. Airborne Collision Avoidance System. In *Proc. 9th Australian Workshop on Safety Critical Systems and Software - Volume 47*, SCS '04, pages 97–110, 2004.
27. M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2002.