MODELING CONFLICTS RESOLUTION OF UNMANNED AIRCRAFT SYSTEM USING A LIGHTWEIGHT DURATION CALCULUS

Diogo Branquinho Ramos [1,2], Rovedy Aparecida Busquim e Silva [1], Inaldo Capistrano Costa [1,3], Emilia M. Colonese [1] and José Maria Parente de Oliveira [1]

[1] Aeronautics Institute of Technology (ITA)

Pç. Mal. Eduardo Gomes, 50, São José dos Campos, SP, Brazil

[2] National Institute for Space Research (INPE),

Av. dos Astronautas, 1758, São José dos Campos, SP, Brazil

[3] Federal University of Maranhão,

Av. Dr. Anselmo, 2008 Campus VII, Codó, MA, Brazil

Abstract

Over the last two decades, an interesting area of Brazilian military and civil sectors is the Unmanned Aircraft Vehicle (UAV) development. This article tackles the modeling of conflicts resolution of Unmanned Aircraft System (UAS) using a lightweight Duration Calculus (DC) to verify if the temporal specification and design of the system is correct and to ensure formally that the system implementation meets all its requirements. Moreover, the article proposes a formal modeling (using DC) of a conflicts resolutions set of rules, adapted from Free Flight concept in Communications, Navigation and Surveillance/Air Traffic Management (CNS/ATM). In the adapted approach to UAS, each UAV is surrounded by an imaginary space of two cylinders, which form, respectively, the protected zone and the alert zone. The major contribution of this article is structuring a new scenario application of the conflicts resolution to UAS through formal modeling, using the DC technique to confirm that the models could be implemented without deadlocks and unreachable states, as well as with satisfaction of temporal restrictions. Furthermore, this work uses the state-of-the-art practices in formal methods, including a model checking tool to ensuring correct real-time requirements specification of a real-time critical system.

Introduction

It became extremely important for the Brazilian military and civil sectors the deployment of Unmanned Aircraft Vehicle (UAV). This vehicle type can be employed in military and civil recognition and surveillance missions, data link, natural resources, energy and oil pipelines monitoring, border security, public safety among others [1]. The authors have developed features related to low cost missions of UAV, in order to reduce time between missions, to provide fast conflict resolutions and graceful maintenance.

In this context some missions require that UAV set their procedures at the same time, acting as UAS, where each UAV performs a specific function in the mission context. For example, a first UAV performs the data link functionality, a second UAV follows a moving target and a third UAV approach (attack or intercept) the target.

Hence, trajectories conflicts must be rapidly solved avoiding collisions between UAVs operating in a same region, ensuring that each UAV is able to fulfill the "Sense-and-Avoid" (S&A) [2], i.e., monitor all the surroundings of an UAV to maintain the S&A of a collision. To ensure that critical systems with real-time characteristics operate safely, it is necessary to establish a rigorous requirements specification.

Real-time Systems (RTS) need to react to certain input stimuli within given time bounds. There are many embedded safety-critical applications and each requires real-time specification techniques.

The focus of this paper is on the formal modeling of the conflicts rules resolutions for UAS. Furthermore, we suggest a set of conflicts rules resolutions adapted from the Free Flight Concept in Communications, Navigation and Surveillance/Air Traffic Management (CNS/ATM), but with centralized control in an UAS Control Station, the Duration Calculus (DC) application in formal

978-1-61284-798-6/11/$26.00 ©2011 IEEE

modeling requirements and the use of timed automata for formal verification by Model Checking tools.

Formal Methods Application

Duration Calculus

Duration Calculus is a temporal logic for continuous time interval that allows the user to specify properties of real-time system without worrying about implementation. A DC formula describes how state variables and time-dependent observables should behave in time intervals [3]. According to this view it is possible to measure the cumulative duration of states. Thus, RTS are modeled through a collection of state variables or time-dependent observables. The Duration Calculus consists of states assertions, terms and formulas built from a set of symbols. The DC subset of interest is DC Implementables.

DC Implementables is a technique developed in DC for the requirements refinement of systems design. Implementables are certain patterns of DC formulas that are well suited for specifying the behavior of control automata. DC Implementables are patterns defined from the following standard forms:

- Initialization: This pattern indicates that the control automaton is in phase π.

$$\lceil\ \rceil \vee \lceil\pi\rceil; true.$$

- Sequencing: This pattern indicates that when the control automaton is in phase π, it stays in π or moves to other phases.

$$\lceil\pi\rceil \vee \lceil\pi \vee \pi_1 \vee \pi_2 \vee \ldots \vee \pi_n\rceil.$$

- Progress: This pattern indicates that after the control automaton has stayed for θ seconds in phase π and then leaves this phase, progressing.

$$\lceil\pi\rceil \overset{\theta}{\rightarrow} \lceil\neg\pi\rceil.$$

- Synchronization: This pattern indicates that the control automaton has stayed for θ seconds in phase π, together with the assertion φ true and then leaves this phase.

$$\lceil\pi \wedge \varphi\rceil \overset{\theta}{\rightarrow} \lceil\neg\pi\rceil.$$

- Bounded Stability: This pattern expresses that when the control automaton has changed its phase to π with the condition φ being true and the time since this change not exceeding θ seconds, it subsequently stays in π (i.e. π is stable) or it moves to one of the phases $\pi_1 \ldots \pi_n$.

$$\lceil\neg\pi\rceil; \lceil\pi \wedge \varphi\rceil \overset{\theta}{\rightarrow} \lceil\pi \vee \pi_1 \vee \pi_2 \vee \ldots \vee \pi_n\rceil.$$

Timed Automata

A timed automaton is a finite-state automaton extended with a finite collection of real-valued clock variables [4]. Also, timed automaton can be defined as a nondeterministic machine and does not have to make transitions as long as the invariance condition is satisfied. One usual way to force transitions is by using an invariance condition [5]. Bringing the definition into practice, a RTS can be described as a set of timed automata, each one representing the behavior of an autonomous process.

Model Checking Tools

Model checking is one of formal techniques. Model checking is an automated formal technique based on the exploitation of states that can be used to obtain evidence that system security is not busted [5]. The selected tool for model checking is UPPAAL.

UPPAAL is a tool for modeling, simulation and verification of RTS [6]. The system model can be validated and verified through simulations and model checker, ensuring that the model implementation is error-free and that the model dynamic behavior can be analyzed and verified. The main function of the tool is to proof the achievement of the system requirements specification by the model. If any requirement is not satisfied, a counter-example is generated to help detect the error and refine the model.

Related Work

This section presents research works related to conflicts resolution of UAVs and examples of application of Duration Calculus and subsets.

In [2], an avoidance algorithm related with conflict resolution for UAVs is proposed. The algorithm considers UAVs in controlled and uncontrolled airspace and must be able to take over the responsibility of the S&A requirement in events such as a loss of data link. As a result of this research group, a patent to Collision and conflict avoidance system for autonomous UAVs was created [7]. The

approach uses on-board sensors to aid the analysis executed for search imminent conflicts. Two sub-systems, Reactive Avoidance System and Planning Avoidance System, are responsible by the task of avoid collision.

Bertuccelli, Luca F. et al. [8] propose an extension to conflict-free assignment algorithm Consensus Based Bundle Algorithm (CBBA). The extension is intended to handle two realistic multi UAVs operation complications. The algorithm is integrated to a simulation environment that allows checking the feasibility of the work.

The work of Olderog [3], has three formal approaches to the specification of RTS: Duration Calculus, DC Implementables, Timed Automata, and PLC-Automata. The book presents a formal specification and automatic verification of RTS, from requirements down to executable code. The traditional examples used are two safety critical systems: the Gas Burner System [9] and the Generalized Railroad Crossing [5].

In [10], the behavior of an aircraft is modeled as autonomous hybrid automata and the reachability analysis is provided by the tool HyTech, to search for optimal trajectories to be employed. The work conclusion asserts that the results that can be obtained by model checking requires less effort in comparison to finding numerical solutions to complicated sets of non-linear equations. Platzer and Clarke [11] show that the formal verification is feasible for applications as complex as aircraft maneuvers. They used a verification tool to verify safety properties

In this work, we propose joining DC formalisms with Conflicts Resolution of UAVs. The goal is to show that DC can be used to aid in temporal requirements specification, as a formal and simple way to bring up more safety to this important topic. Furthermore, we show a possible use of DC for a scenario based on the aerospace spectrum, to formally checking the conflict resolution of UAVs.

Conflicts Resolution Rules for UAS

With the growing number of UAVs in a single operation, rules definition for solving conflicts in UAS is extremely necessary, as in the Free Flight concept in CNS/ATM to the aircraft flight phase.

In the adapted approach to our UAS, each UAV is surrounded by an imaginary space of two cylinders, which form, respectively, the protected zone and the alert zone. No UAV may enter the protected zone of another one and when a UAV enters the alert zone of another, a protocol for conflict resolution should be performed. Figure 1 shows the alert and protected zones from an UAV and the maximum time for resolving the conflict before UAVs collisions.

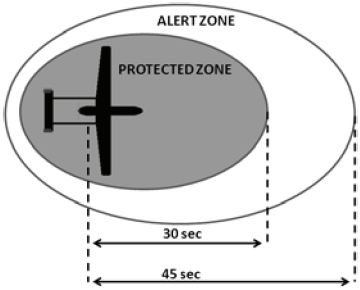

Figure 1. Alert and Protected Zones Demarcation of an UAV

In order to identify the position surrounding the UAV, which a conflict can happens, the alert zone is divided into regions: P (Port Front), S (Starboard Front), and R (Rear), illustrated in Figure 2.

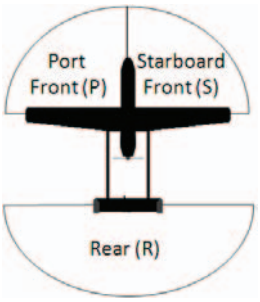

Figure 2. Alert Zone Demarcation of UAV

After the definition of the UAV alert and the protected zones, the protocol for resolving conflicts based on behavior rules, allows an UAV to be able to leave a situation considered critical, i.e. not stay on the alert zone of another UAV. In this study we consider the following Conflict Resolution rules [12]:

- Rule #1 - If UAV A and UAV B are both in the P region of one another, then both UAVs must turn right and then return to their original trajectory;
- Rule #2 - If UAV A and UAV B are both in the S region of one another, then both

UAVs must turn left and then return to their original trajectory; and

- Rule #3 - If UAV A is in the R region of the UAV B and UAV B is in the S or P region of the UAV A, then UAV A must turn in either direction to achieve separation with minimal deviation and then return to its original trajectory.

The Figure 3 illustrates the conflict resolution rules.

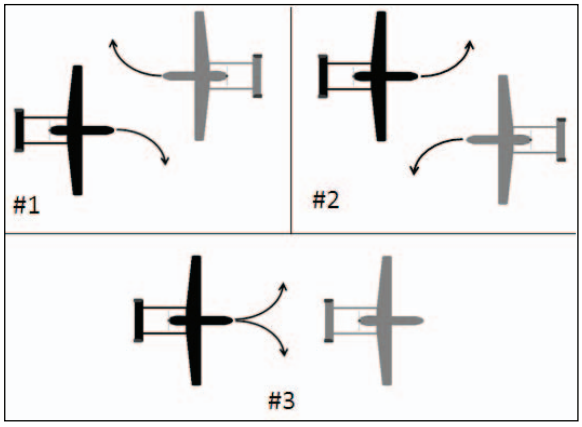

Figure 3. Conflict Resolution Rules

Conflicts Resolution Formal Approach for UAS

The approach proposes automatic real-time requirements verification and is based on DC and extended timed automata as shown in Figure 4. The real-time requirements are specified in DC and in a subset of DC Implementables. The requirements automatic verification is performed using the model checking tool UPPAAL, through timed automata.

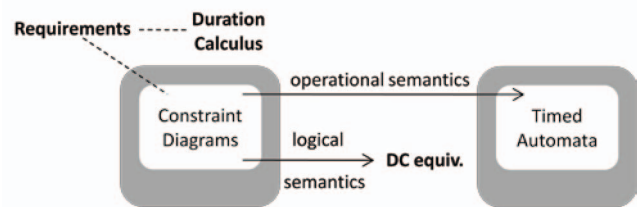

Figure 4. Proposed Approach, Adapted from [2]

The methodology proposed consists of:

1. Specify the real time requirements formally;
2. Check requirements through model checking;
3. Perform a model checking simulation; and
4. Analyze Results via model checking verification.

Specify the Real Time Requirements Formally

The formalization of the real-time requirements is organized into four steps.

Step 1 - DC Specification

The formalization is focused on the UAS security aspect. We begin the work identifying the system observables and the control system iteration description with the environment. The Figure 5 illustrates the control system iteration with the environment, detailing how the controller gets information and how it operates in the environment, this specification corresponds to monitoring only one UAV. At present, four observables are identified, with two inputs (A and P) and with two outputs (S and D).

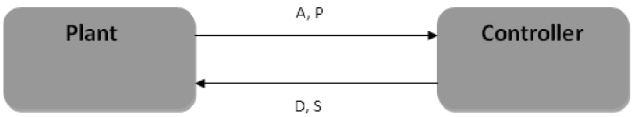

Figure 5. System Iteration with the Environment

The observable *A* describes when an UAV is in the alert zone of another monitored UAV and what the region was overrun (front port, starboard front, rear or empty). The observable *P* describes when there is an UAV in the protected zone of another monitored. The observable *D* describes when a monitored UAV changes its direction (left, right or keep), and *S* describes when a monitored UAV should change its speed.

$A: Time \rightarrow \{port\ front, starboard\ front, rear, empty\}$

$P: Time \rightarrow \{0,1\}$

$D: Time \rightarrow \{left, right, keep\}$

$S: Time \rightarrow \{increase, decrease, keep\}$

The observable *P* has logical states: true (1) and false (0). Its mere representation (*P*) is characterized as true, and its representation with the negation symbol ($\neg P$) is characterized as false.

Note that via the four values of the observable *A* we have abstracted from further details of the plan, the exact position of the UAV in alert zone, as well as for the observable *D*. We will use the following abbreviations:

Ap(t) stands for $A(t) = port\ front$

As(t) stands for $A(t) = starboard\ front$

Ar(t) stands for *A(t) = rear*

Ae(t) stands for *A(t) = empty*

Dl(t) stands for *D(t) = left*

Dr(t) stands for *D(t) = right*

Dk(t) stands for *D(t) = keep*

Si(t) stands for *S(t) = increase*

Sd(t) stands for *S(t) = decrease*

Sk(t) stands for *S(t) = keep*

The safety critical state *C* describes when the UAV will collide with another UAV, when an UAV is in the alert zone *A* (in any region, port front, starboard front or rear) and in the protected area *P* of the monitored UAV. This state can be formalized with the Boolean expression:

$$C \overset{\text{def}}{\Leftrightarrow} \boxtimes\big((\lceil Ap\rceil \vee \lceil As\rceil \vee \lceil Ar\rceil) \wedge \lceil P\rceil\big), where$$

$$C: Time \rightarrow \{0,1\}$$

In this expression, when the *C* has logical states as 1, it means that the safety critical state is true and when *C* has logical states as 0, false. The expression means that for all intervals in which it occurs *Ap*, *As*, or *Ar* and *P*, *C* is true.

The mere representation (*C*) is characterized as true, and its representation with the negation symbol (¬ *C*) is characterized as false.

The real time requirement of the UAV is that it has a period of 15 time units to exit the alert zone (*P*, *S*, or *R* regions) of another monitored UAV.

$$Req \overset{def}{\Leftrightarrow} \boxtimes\ (\ \lceil Ap\rceil \vee \lceil As\rceil \vee \lceil Ar\rceil; \lceil Ae\rceil\ ) \Longrightarrow \ell \leq 15$$

Where, *Req* is defined as for all interval in which *Ap*, *As*, or *Ar* occurs followed by *Ae* with ℓ time less than or equal to 15.

Step 2 - Design

Based on high-level requirement set in the previous step, the controller is designed. The authors have decided to introduce in the design model a real-time constraint that appeared to be easier to implement, which together imply the requirement *Req*:

- The controller can apply the rule of conflict resolution #1 when an UAV exists in the region *P*.

$$Des1 \overset{def}{\Leftrightarrow} \boxtimes\ (\ \lceil Ap\rceil; \lceil Dr\rceil; \lceil Ae\rceil\ ) \Longrightarrow \ell \leq 15$$

Where, *Des1* is defined as for all intervals in which *Ap* occurs followed by *Dr*, and followed by *Ae* with ℓ time less than or equal to 15.

- The controller can apply the rule of conflict resolution #2 when an UAV exists in the region *S*.

$$Des2 \overset{def}{\Leftrightarrow} \boxtimes\ (\ \lceil As\rceil; \lceil Dl\rceil; \lceil Ae\rceil\ ) \Longrightarrow \ell \leq 15$$

- The controller can apply the rule of conflict resolution #3 when an UAV exists in the region *R*

$$Des3 \overset{def}{\Leftrightarrow} \boxtimes\ (\ \lceil Ar\rceil; \lceil Si\rceil; \lceil Ae\rceil\ ) \Longrightarrow \ell \leq 15$$

Step 3 - Implementables

Then a subset of the Duration Calculus that is closer to the implementation level is created, the so-called DC Implementables.

We model a conflict resolution as a system of five control automata based on the initial DC specification, represented by the following state variables:

- A state variable *Ctr* ranging over {waiting, verify alert, conflict resolution, verify conflict} representing the *controller*;
- A Boolean state variable *P* representing *protected zone*;
- A state variable *A* ranging over {*Ae, As, Ap, Ar*} representing the *alert zone regions*;
- A state variable *D* ranging over {*Dk, Dr, Dl*} representing the *direction;*
- A state variable *S* ranging over {*Sk, Si, Sd*} representing the *speedy*.

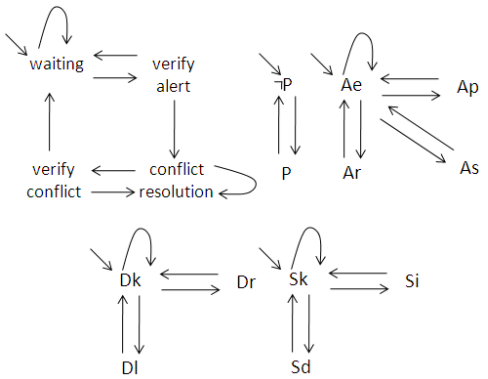

Figure 6. *Ctr, P, A, D* and *S* Transition Diagrams

Figure 6 shows the untimed transitions behavior of the state variables *Ctr, P, A, D* and *S*.

Step 4 - Implementables with Time-Dependent Behavior

To insert a time-dependent behavior into DC Implementables of the control automata *Ctr*, *P*, *A*, *D* and *S*, a global variable ε representing the parameter for time reaction is created and the DC Implementables are therefore described as follows.

$Init1$: $\lceil\ \rceil \vee \lceil \neg P \rceil; true$,

$Init2$: $\lceil\ \rceil \vee \lceil Ae \rceil; true$,

$Init3$: $\lceil\ \rceil \vee \lceil Dk \rceil; true$,

$Init4$: $\lceil\ \rceil \vee \lceil Sk \rceil; true$,

$Init5$: $\lceil\ \rceil \vee \lceil waiting \rceil; true$,

$Seq1$: $\lceil waiting \rceil \longrightarrow \lceil verify\ alert \rceil$,

$Seq2$: $\lceil verify\ alert \rceil \longrightarrow \lceil waiting \vee conf\ res \rceil$,

$Seq3$: $\lceil conf\ res \rceil \longrightarrow \lceil verify\ conflict \vee conf\ res \rceil$,

$Seq4$: $\lceil verify\ conflict \rceil \longrightarrow \lceil conf\ res \vee waiting \rceil$,

$Seq5$: $\lceil Ae \rceil \longrightarrow \lceil As \vee Ap \vee Ar \vee Ae \rceil$,

$Seq6$: $\lceil As \rceil \longrightarrow \lceil Ae \rceil$,

$Seq7$: $\lceil Ap \rceil \longrightarrow \lceil Ae \rceil$,

$Seq8$: $\lceil Ar \rceil \longrightarrow \lceil Ae \rceil$,

$Seq9$: $\lceil Dk \rceil \longrightarrow \lceil Dr \vee Dl \vee Dk \rceil$,

$Seq10$: $\lceil Dr \rceil \longrightarrow \lceil Dk \rceil$,

$Seq11$: $\lceil Dl \rceil \longrightarrow \lceil Dk \rceil$,

$Seq12$: $\lceil Sk \rceil \longrightarrow \lceil Si \vee Sd \vee Sk \rceil$,

$Seq13$: $\lceil Si \rceil \longrightarrow \lceil Sk \rceil$,

$Seq14$: $\lceil Sd \rceil \longrightarrow \lceil Sk \rceil$,

$Syn1$: $\lceil waiting \wedge Ae \rceil \xrightarrow{\epsilon} \lceil Dk \wedge Sk \rceil$,

$Syn2$: $\lceil conf\ res \rceil \xrightarrow{\epsilon} \lceil Dr \rceil$,

$Syn3$: $\lceil conf\ res \wedge As \rceil \xrightarrow{\epsilon} \lceil Dl \rceil$,

$Syn4$: $\lceil conf\ res \wedge Ar \rceil \xrightarrow{\epsilon} \lceil Si \rceil$,

$Stab1Init$: $\lceil waiting \wedge Ae \wedge Dk \wedge Sk \wedge \neg P \rceil \longrightarrow_0 \lceil waiting \rceil$,

$Stab2$: $\lceil waiting \rceil; \lceil verify\ alert\ \wedge \neg Ae \rceil \longrightarrow \lceil conf\ res \rceil$,

$Stab3$: $\lceil conf\ res \wedge Ap \rceil; \lceil Dr \rceil \xrightarrow{\leq 15} \lceil Ae \rceil$,

$Stab4$: $\lceil conf\ res \wedge As \rceil; \lceil Dl \rceil \xrightarrow{\leq 15} \lceil Ae \rceil$,

$Stab5$: $\lceil conf\ res \wedge Ar \rceil; \lceil Si \rceil \xrightarrow{\leq 15} \lceil Ae \rceil$,

$Stab6$: $\lceil conf\ res \rceil; \lceil verify\ conflict\ \wedge Aen \wedge Dk \wedge Sk \rceil \longrightarrow \lceil waiting \rceil$.

Check Requirements via Model Checking

The UPPAAL tool was used to model the conflict resolution system in conformity with the Duration Calculus specification, in order to formally verify and simulate the model.

The UPPAAL Declarations are defined to model the communication channels among the system components and their global variables. Below are the model statements and their association with the specified calculation period:

```
// UAS : Unmanned Aerial System
// Declarations Section

clock time;        // event time control
const int N = 2;  // uav qtd (N)
bool alert[N], original_trajectory[N];  // UAV states
int region[N];

// A state alert "region" ranging over {Ae, As, Ap, Ar}
// representing the alert zone regions:
// region == 1  --> Ap(t) stands for A(t) = portfront
// region == 2  --> As(t) stands for A(t) = starboardfront
// region == 3  --> Ar(t) stands for A(t) = rear
// region == 4  --> Ae(t) stands for A(t) = empty

typedef int[0,N-1] id_uav;       // uav identification
broadcast chan  collectData; // data acquisition channel
chan turnLeft, turnRight;      // direction channel
```

These variables are used by the automata Controller as guard conditions and transitions control in the UPPAAL model, consisting of two components modeled as timed automata (the controller and the UAV).

Controller

The Controller component composes the UAS, was translated in one timed automaton as shown in Figure 7. It was implemented to control the required functionalities of keeping two UAVs protected, according with rules described in the section of Conflicts Resolution Rules for UAS.

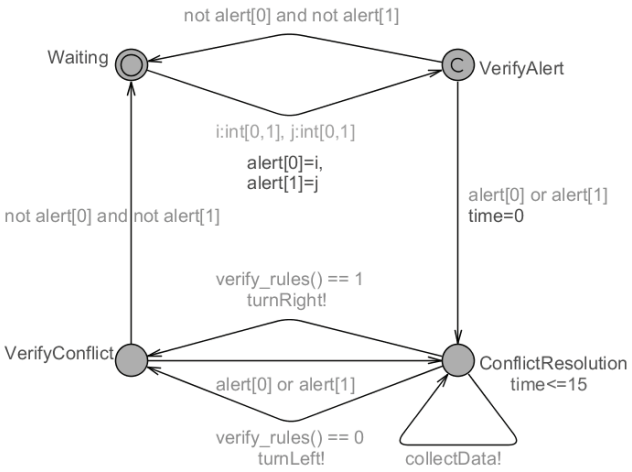

Figure 7. UAS Controller Automaton

It can be noticed that the Controller is modeled according to the corresponding states defined in the CD Implementable: *Waiting*, VerifyAlert, VerifyConflict and ConflictResolution.

UAV

The UAV component was also designed based on the system requirements as shown in Figure 8.

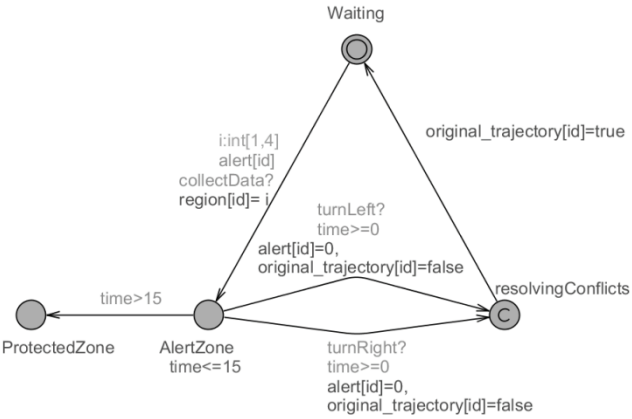

Figure 8. UAV Automaton

The UAV automaton can be instantiated in several UAVs automata, according to the desired simulation. Our initial model was instantiated with two UAVs.

Perform a Model Checking Simulation

The automatic verification of UAS has used the model checking UPPAAL tool. The simulation ensures the implementation of the model without path errors. The system prototype model – UAS was tested to evaluate system failures and behavior correction [13].

Figure 9 shows a partial simulation view of the UAS execution, where the correct behavior of the prototype can be verified by model inspection.

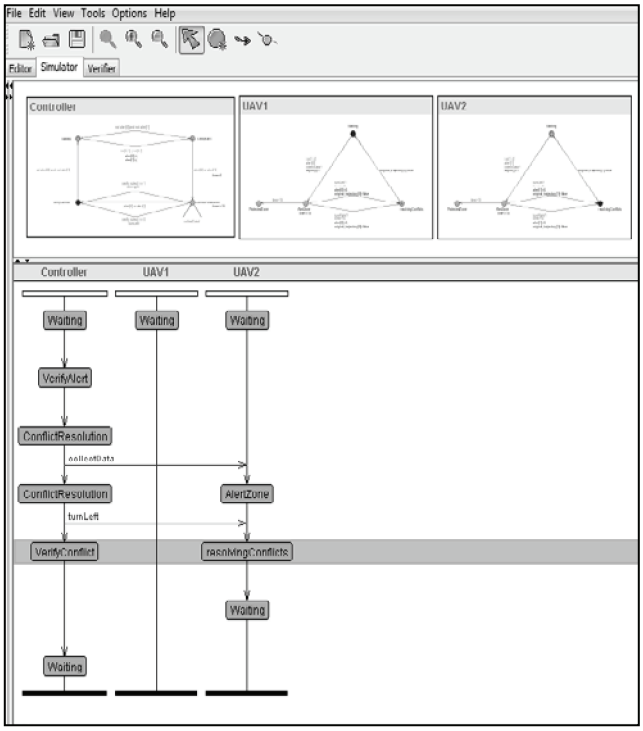

Figure 9. UAS Simulation

In this execution the sequence in which an alert state is identified is clearly presented.

The alert is associated with the UAV2, which receives a message from the Controller and changes its trajectory by making a critical move to the left, in agreement with the specified rules.

Analyze Results Via Model Checking Verification

Also, UPPAAL can verify reachability, liveness and safety properties on UAS. According to Lamport [14] a liveness property asserts that something good eventually will happen, and a safety property that something bad should never happen. Reachable is a property that can eventually hold. In this work, only reachability and safety properties are verified along with system interoperability. The following set of properties was translated to CTL [15], and inserted into the UPPAAL for verification.

- P1: Checking absence deadlock:

```
A[] not deadlock;
```

- P2, P3: Checking the safety critical *C* described in the DC Specification section:

```
A[] not UAV1.ProtectedZone; // UAV1 does not reach the protected zone of UAV2

A[] not UAV2.ProtectedZone; // UAV2 does not reach the protected zone of UAV1
```

- P4: Checking the requirement *Req* described in the section of Conflicts Resolution Formal Approach for UAS.

```
A[] UAV1.AlertZone imply (time<=15) // UAV1 has a period of 15 time units to exit the alert zone
```

- P5: Checking the DC Implementable *Seq-2*. The possible states that the controller can reach from state *verifyalert* are *waiting* and *conflict resolution*:

```
Controller.VerifyAlert-->
(Controller.ConflictResolution or
Controller.Waiting)
```

- P6: Checking the DC Implementable *Syn-2*. The possible states that the controller can reach from state *VerifyAlert* are *waiting* and *conflict resolution*:

```
Controller.ConflictResolution and
UAV1.AlertZone -->
(Controller.ConflictResolution or
UAV1.resolvingConflicts)
```

- P7: Checking the DC Implementable *Stab-2*. The controller must resolve a conflict if the UAV is in one of the alert zone regions.

```
Controller.VerifyAlert and
UAV1.AlertZone -->
Controller.ConflictResolution
```

Figure 10 shows that these properties have been satisfied.

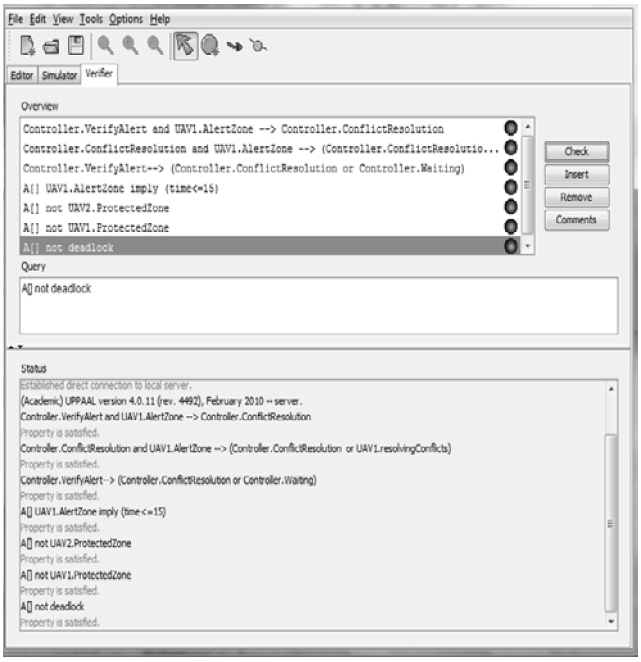

Figure 10. Model Checking Results Verification

Conclusion

The rules from Free Flight concept in CNS/ATM had enabled a new scenario application of conflicts resolution for UAS using formal modeling.

In this work a formal modeling approach was adopted, and the results had shown that it is feasible and efficient enough for ensuring correct real-time requirements specification of a real-time critical system.

The formal specification of real-time requirements with Duration Calculus, had allowed the expansion of the potential application of Duration Calculus in other knowledge areas.

Also, developing the implementables had allowed earlier planning of problems resolution to meet the specified formal requirements.

The proposed approach provides the basis for implementation of embedded devices or monitoring systems to solve conflicting routes of UAV in a UAS. Given the worldwide growing use of this kind of aircraft, solving conflicts can be the proper measure for mission accomplishment or for safer operation.

For future work, the authors suggest the creation of new rules of conflict resolution to be formally verified and validated.

References

[1] Ramos, D., Loubach, D., Cunha, A., 2010, Developing a Distributed Real-Time Monitoring System to Track UAVs. Aerospace and Electronic Systems Magazine, IEEE 25(9), pp. 18-25.

[2] Van Tooren J., Heni, M. A., Knoll, J. B., 2007, Development of an Autonomous Avoidance Algorithm for Uavs in General Airspace. Proceedings of First CEAS European Air and Space Conference.

[3] Olderog, E.R., H. Dierks, 2008, Real-Time Systems: Formal Specification and Automatic Verification.

[4] R. Alur and D. Dill, 1994, A Theory of Timed Automata, Theoretical Computer Science, vol. 126, pp. 183-235.

[5] Heitmeyer C., N. Lynch., 1994, The generalized railroad crossing. In IEEE Real-Time Systems Symposium, IEEE Computer Society Press, pp. 120–131.

[6] Behrmann, G., David, A., and Larsen, K. G., 2004, A tutorial on UPPAAL, In School on Formal Methods for the Design of Computer, Communication and Software Systems.

[7] Van Tooren J., Heni, M. A., Knoll, J. B., 2010, Collision and Conflict Avoidance System for Autonomous Unmanned Air Vehicles (UAVS). United Stated Patent No US 7737878 B2.

[8] Bertuccelli, Luca F. et al., 2009, Real-Time Multi-UAV Task Assignment in Dynamic and Uncertain Environments. AIAA Guidance, Navigation, and Control Conference, Chicago, Illinois, pp .1-16.

[9] Ravn A. P., H. Rischel, K.M. Hansen, 1993, Specifying and Verifying Requirements of Real-Time Systems, IEEE Transactions on Software Engineering, pp. 19:41–55.

[10] Massink, M.; De Francesco, N., 2001, Modeling Free Flight with Collision Avoidance, Engineering of Complex Computer Systems, pp 270 – 279.

[11] Platzer A., Clarke E. M., 2009, Formal Verification of Curved Flight Collision Avoidance Maneuvers: A Case Study, FM 2009: Formal Methods Lecture Notes in Computer Science, Volume 5850/2009, pp. 547-562.

[12] Hilburn, B., 2010, Air Traffic Control (ATC) Aspects of Free Flight, Online, http://www.freeflightatm.org.

[13] Hessel, A., Larsen, K.G., Mikucionis, M., Nielsen, B., Pettersson, P., Skou, A., 2008, Testing Real-Time Systems Using Uppaal. In: Formal Methods and Testing, LNCS, Springer, pp. 77-117.

[14] Lamport, L., 1980, “Sometimes" Is Sometimes “Not Never": on the Temporal Logic of programs, In: Proc. 7th ACM Symp. on Principles of Programming Languages, pp. 164-185.

[15] Behrmann, G., David, R., Larsen, K.G., 2004, A tutorial on UPPAAL, Springer, pp. 200-236.