Formal Verification Tool Evaluation For Unmanned Aircraft Containing Complex Functions

Heber Herencia-Zapana
General Electric Global Research Center
Niskayuna, NY
heber.herencia-zapana@ge.com

James Lopez
General Electric Global Research Center
Niskayuna, NY
lopezj@ge.com

Glen Gallagher
GE Aviation Systems
Pinellas Park, FL
Gallagher@ge.com

Baoluo Meng
General Electric Global Research Center
Niskayuna, NY
baoluo.meng@ge.com

Cameron Patterson
Electrical and Computer Engineering
Virginia Tech
Blacksburg, VA
cdp@vt.edu

Lakshman Maalolan
Electrical and computer Engineering
Virginia Tech
Blacksburg, VA
tmlakshman@vt.edu

Abstract **The expected proliferation of UAS in the NAS requires technologies that ensure safe operation. There is significant interest from industry and civil aviation authorities to have a standard practice to enable flight operations for UAS containing flight safety critical functions which are too costly to certify. Developing a certification path for these UAS technologies could advance safety of UAS operating in the NAS. In response to this need ASTM released standard F3269-17 in 2018. This standard proposes a run-time assurance architecture whereby an untrusted or non-pedigreed and therefore non-certified flight safety critical function (complex function) can be included in a UAS avionics system that can be certified. GE Aviation is developing an avionics solution intended for safe operation of UAS. As part of ensuring safe operation of UAS GE Aviation's avionics implements a runtime safety assurance (RTA) system that follows the guidelines laid out in the ASTM F3269-17 standard.**

Formal methods-based verification and validation (V&V) tools hold great promise for addressing the exploding cost of performing V&V on flight safety critical systems that include software. However, there are very few examples demonstrating a side- by-side comparison of the traditional V&V approach and a V&V approach where formal methods-based tools are used at appropriate steps in the process.

This paper presents a side-by-side comparison of a complete V&V process for the RTA using both traditional and formal methods-based V&V and shows the benefits of formal tools applied at various early stages of the V&V process. More specifically this paper shows a comparison for the generation of the following evidence for the RTA: Requirements analysis, test case generation, and prof that requirements are fully implemented by the select sub-systems and/or components architecture.

Keywords—UAS, Certification, F3269-17, Formal methods, V&V process, safety critical systems.

I. INTRODUCTION

The expected proliferation of UAS in the NAS requires technologies that ensure safe operation. There is significant interest from industry and civil aviation authorities to have a standard practice to enable flight operations for UAS containing flight safety critical functions which either cannot be certified (e.g., non-deterministic software) or are too costly to certify (e.g., open source autopilot). In response to this need ASTM released standard F3269-17 in 2018 [2], "Standard Practice for Methods to Safely Bound Flight Behavior of Unmanned Aircraft Systems Containing Complex Functions". This standard proposes a run time safety assurance architecture whereby an untrusted or non-pedigreed and therefore non-certified flight safety critical function (complex function) can be included in a UAS avionics system that can be certified. The standard proposes that a pedigreed safety monitor have the authority to take control of a vehicle management system (VMS) away from the untrusted complex function in the event that the complex function attempts to send a command to the VMS that violates a pre-defined safety policy.

GE is implementing a Run-Time Assurance (RTA) system on its M100 UAS Avionics compute platform. GE's RTA implements F3269-17 to bound the behavior of an open source autopilot. For the certification, it is necessary to generate evidence that RTA deliver functionality in accordance with the guidance of F3269-17. GE is working to perform a traditional V&V process on the RTA system for the purpose of generating sufficient evidence to obtain an FAA Part 107 waiver and eventually an airworthiness approval.

Formal methods-based verification and validation (V&V) tools hold great promise for addressing the exploding cost of performing V&V on flight safety critical systems that include

978-1-7281-9825-5/20/$31.00 ©2020 IEEE

software. Many examples exist in the industry and in the literature where formal methods tools have been applied to automate various steps in the V&V process (e.g., formal analysis of formalized requirements and formalized models and auto-generation of requirements-based test cases), however there are very few examples demonstrating a side-by-side comparison of the traditional V&V approach and a V&V approach where formal methods-based tools are used at appropriate steps in the process.

This paper presents a side-by-side comparison of a complete V&V process for the RTA using both traditional and formal methods-based V&V and shows the benefits of formal tools applied at various early stages of the V&V process. More specifically this paper shows a comparison for getting the benefits of having requirements for the RTA.

A requirement precisely expresses what is needed to be implemented and what we expect to get from a system. The requirements contain the behavior, attributes and properties of the system. Therefore, the main benefits of the requirements are: First, to create a list of terms that are going to be used in the description of the system. Second, to be free of any ambiguities. That is to say, all the stakeholders should understand requirements in the same way, and they are understood by all stakeholders. Third, software artifact traceability, which is the ability to describe and follow the lifecycle of an artifact (requirements, code, tests, models, reports, plans, etc.) developed during the software lifecycle. Four, to provide a clear goal in the software implementation phase. Five, to provide means of verifying the compliance of the implementation.

The requirements benefits are going to be compared using the traditional and formal approach. But, in order to study the comparison of the benefits of the requirements when they are constructed in a traditional and formal approach, it is necessary to have a baseline for where to begin the comparison study. Informal requirements are the basis for the construction of the baseline. The comparison is explained in the following sections: Section II, Requirements base line capture, Section III, Requirement capture and management comparison, Section IV, Test cases generation comparison, Section V, Requirement implementation comparison, and Section VI, Assurance case generation comparison.

II. Requirements Base Line Capture

A side-by-side comparison of a complete V&V process on a relevant real-world flight safety critical system using both traditional and formal methods-based V&V will show the benefits of formal tools applied at various early stages of the V&V process. This section presents a subsystem of the RTA, namely a geofence boundary monitoring subsystem, which will be used for the side-by-side comparison.

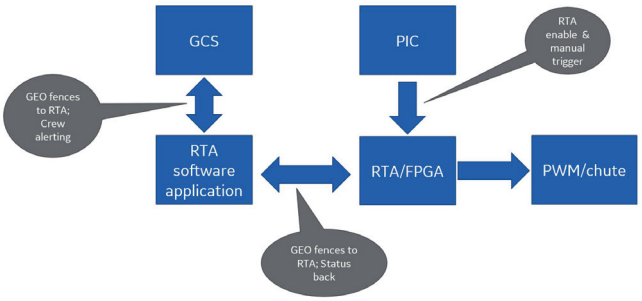

Figure 1

Figure 1 depicts the overall RTA system block diagram as well as the specific RTA subsystem selected for this study. The selected subsystem is a Xilinx RTA/FPGA which hosts the VHDL implementations of the safety monitors. There are 3 safety checks hosted on the RTA/FPGA:

1. Vehicle is inside/outside geofence
2. Conformance with performance envelope (pitch, roll, yaw rate limits)
3. GPS check (comparison of GPS location data from 2 antennas)

The main inputs to the RTA/FPGA are the parameters that define the geofenced boundaries and current performance parameters values as well as the current GPS position (and position uncertainty) of the vehicle (this data is provided to the RTA/FPGA by the Inertial Navigation System). For the comparison study we selected the safety monitor that checks that the vehicle is inside or outside the geofenced boundary.

GE Aviation has developed a set of requirements for the RTA system. We have selected the subset of requirements that are allocated to the monitors. An example of RTA requirements allocated to the geofence monitor are listed in Table 1.

Requirement Number	Requirement Description
RTA-HW-56	The RTA block shall use the following algorithm for determining inside a polygon: - For each edge defined by x1,y1 to x2,y2, including last point to first point: Count by one if : ▪ (y1 <= y < y2) AND ((x1 * y2) – (x2*y1) > 0) OR ▪ (y2 < y <= y1) AND ((x1 * y2) – (x2*y1) < 0) - If count is odd then inside; otherwise outside.
RTA-HW-61	The RTA block shall declare a geofence trip if any of the following is detected for 3 consecutive position inputs: - Any exclusive fence is violated - No inclusive fence is satisfied and there is at least one inclusive fence - The safety fence is violated

Table 1

A. Functional Requirement Format

Functional architecture is an architecture model that identifies system functions and their interactions. It defines how the functions will operate together to perform the system mission. A functional specification in system engineering is a document that specifies the functions that a system or component must perform. It typically describes what is needed by the system user as well as required properties of inputs and outputs. On more complex systems, such as RTA, multiple levels of functional specifications will typically be nested within each other.

To set the baseline for comparison, the requirements need to be in a special format. The format should enable requirement analysis, management and allow the mapping to a functional architecture. The main property of this format is that the requirements need to describe explicitly the inputs, outputs and a function connecting the inputs and outputs. Table 2 shows the required format of the requirement RTA-HW-61.

Input	Output	Requirements RTSA-HW-61 as functions	Functions call
Nfences: set of fences Position:p1,p2,p3	GeofenceNoTrip	IF Exists p:{p1,p2,p3}: fencesStatus (Nfences, p) = GeoNoTrip THEN GeofenceNoTrip	Req RTSA-HW-61 expressed as function fencesStatus
Nfences Position p1,p2, p3	GeofenceTrip	IF For all p:{p1,p2,p3} : fencesStatus (Nfences,p1) = GeoTrip and fencesStatus (Nfences,p2) = GeoTrip and fencesStatus (Nfences,p3) = GeoTrip THEN GeofenceTrip	
Nfences: set of fences Position:p	GeoTrip GeoNotrip	IF [fencesStatusEX (Nfences,p) = ExcViolated] OR [fencesStatusIN (Nfences,p) = IncViolated] THEN fencesStatus (Nfences,p) = GeoTrip	Function fencesStatus in terms of functions fencesStatausEX and fencesStatusIN
Exclusive fence position	ExcVIolated	IF Exists fE : Nfences: geofenceStatus(fE , p) = GeoViolated THEN fencesStatusEX (Nfences,p) = ExcVIolated	fenceStatusEX in terms of geofenceStatus
Inclusive fences position	IncViolated	IF For all fI : Nfences: geofencesStatus(fI1 , p)=GeoViolated and ⋮ geofencesStatus(fIN , p)=GeoViolated THEN fencesStatusIN (Nfences,p) = IncViolated	fencesStatusIN in terms of geofenceStatus

Table 2: Req RTSA-HW-61 as Req hierarchy

Table 2

III. Requirements Capture and Management Comparison

This section describes the results of the side-by-side comparison in two parts. First, we explain the advantages and disadvantages of the traditional and formal methods approach when the RTA requirements are captured using these two methodologies. Second, we present the metrics of the comparison of man-hours required to perform each step in the traditional and formal approaches to RTA requirements capture, analysis and requirements-based test case generation.

In order to perform the side-by-side comparison on RTA requirements baseline using both traditional and formal methods, a set of comparison criteria needs to be defined. We first explain in more detail the set of comparison criteria, which are: glossary of terms, ambiguous requirements identification, traceability and requirements analysis.

Often, customers and engineers fail to communicate clearly with each other because they may come from different disciplines and do not understand technical terms in the same way. This can lead to confusion and severe miscommunication, and an important task during the requirements management and analysis phase is to ensure that both parties have a precise understanding of the requirements. It is important to be consistent in using words. It is necessary to make a **glossary of the terms** that are going to be used right at the start, ensure all stakeholders have a copy, and stick to them consistently, especially when requirements are captured.

Ambiguous requirement is a requirement that can be interpreted in more than one way and it is not clear which is the intended interpretation. Some factors that cause requirement ambiguities are: A context is assumed, but it is not captured. Terms are not used consistently throughout the requirements.

One of the most important components of requirement management is **traceability**. Tracing allows us to understand why requirements exist and the impact of change. Every time you are looking at changing a requirement, traceability helps to understand the impact it will have to other requirements.

Requirement analysis is critical to the success or failure of a system or software project. Requirements analysis determines whether the requirements are incomplete, independent or contradictory and then resolving these issues [2,3,4].

The comparison between the traditional approach using DOORS® and the formal methods approach using Analysis of Semantic Specifications and Efficient generation of Requirements based Tests (ASSERT™) will be done using a baseline set of requirements and comparison criteria. Let us first describe the main characteristics of DOORS® and ASSERT™.

DOORS® is a development project information and requirements management tool. Project information may consist of designs, tests, standards, as well as the relationships between these. Requirements are conceived to be pieces of text with attributes, for instance attributes such as safety and reliability. The strength of the tool is revealed when the user creates traceability links using attributes. Typically, traceability is done between one requirement and another, but the tool can equally manage traces between system requirements, design elements, tests, data dictionary and other items of project information. The user is free to create links of different types to indicate different logical relationships [6].

The ASSERT™ tool suite has been developed due to increasing complexity and costs of systems development. ASSERT™ consists of four components. The first component is the Requirements Capture Environment which includes GE Requirements Language and SADL grammar [5,7]. This is contained within the Eclipse IDE. The second component is the Requirements Analysis Tool which analyses the written requirements [8]. It provides completeness checks and results to the user. The third component is the Automated Test Generation which auto-generates test cases and procedures to verify the written requirements, via the sub-components ATCG (automated test case generator) and ATPG (automated test procedure generator). Finally, Automated Test Procedure Translation produces test scripts that can be run on the target test environment [1,3].

This work is being performed under NASA System Wide Safety contract #80NSSC19M0239.

A. *DOORS® vs. ASSERT™ Advantage And Disadvantages Comparison*

We conducted an experiment whereby a GE Aviation Systems V&V engineer with extensive experience with DOORS® but little knowledge of formal methods and ASSERT™ performed an experiment using both DOORS® and ASSERT™. The main task of this experiment was to capture the RTA baseline requirements and generate a set of requirements-based test cases.

The ontology is roughly equivalent to a functional architecture and defines both the external functional interfaces across the boundary of the system, but also the internal interfaces within the boundary of the system. The process of capturing the RTA ontology and requirements using DOORS® and ASSERT™ is as follows: The requirements and ontology, which capture the glossary of terms are generated and maintained in the requirements database in DOORS®. This facilitates traceability with requirements, ontology as well as traceability to verification test cases. Then scripts are used to export the ontology and requirements into properly formatted files (DXL scripts in DOORS®). ASSERT™ uses these files to create the model of the system which it uses for capturing requirements errors. Any errors found in the tool are manually fixed in the requirements database and re-exported. Figure 2 depicts the ontology captured in DOORS® and Figure 3 depicts the ontology translated to ASSERT™.

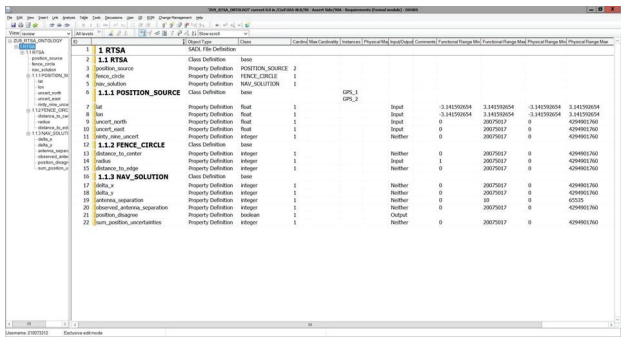

Figure 2

```
28 nav_solution describes RTSA with values of type NAV_SOLUTION.
29 nav_solution of RTSA only has values of type NAV_SOLUTION.
30 nav_solution of RTSA has exactly 1 values.
```

Figure 3

The findings of this experiment are: DOORS® tool does not provide capabilities for identification of ambiguous requirements; this needs to be done manually. But ASSERT™ supports ambiguous requirements checks through the constant check that the requirement uses the terms consistently from the ontology.

Let's illustrate this with an example: The requirement RTA-HLR-8 is captured in both DOORS® and ASSERT™. DOORS® does not identify requirement ambiguity errors, but ASSERT™ identifies two errors related to ambiguity, which are: variable not defined and type check error.

```
22 Requirement RTSA-HLR-8:
23 RTSA shall
24 set Position_Disagreement of NAV_SOLUTION to true
25 when
26 sum_position_uncertainties of GPS_Solution < observed_antenna_separation of GPS_Solution.
```

Figure 4

More specifically, ASSERT™ shows an error in line 24 (Figure 4), which is "Position_disagreement" is not defined in the ontology. ASSERT™ also shows in line 26, a type check error which is the predicate "sum_position_uncertainties" does not apply to the variable "GPS_Solution".

Requirement analysis capabilities of ASSERT™ can be used to analyze requirements as soon as they are written, without the need for lower-level requirements, annotations, properties or code. ASSERT™ for example can also be used to analyze an incomplete set of requirements, which allows requirements engineers to get meaningful feedback immediately. On the other hand DOORS® does not offer automatic requirement analysis. Table 3 summarize the comparison criteria results.

	Comparison criteria			
	Requirement analysis	***Glossary of terms***	***Ambiguous requirement identification***	***Traceability***
DOORS®	Manual	Support	Manual	Automatic
ASSERT™	Automatic	Support	Automatic	Automatic

Table 3

B. *Man-hours Metrics for Comparison of Traditional vs. Formal Capture of RTA Requirements*

We conducted an experiment whereby a GE Aviation Systems V&V engineer with extensive experience with DOORS® but little knowledge of formal methods and ASSERT™ performed the steps in Figure 5 for both the traditional and formal approach (all except for the last step of the formal approach, auto-generation of requirements-based test cases, which requires a working knowledge of ASSERT™) and tracked the amount of time in man-hours to complete each step.

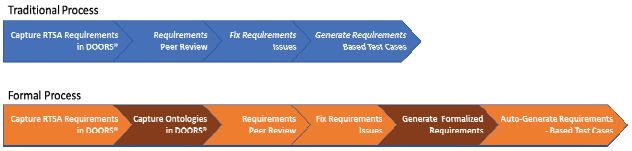

Figure 5: Side-by-side process (traditional and formal) for capturing RTA requirements

Figure 6 shows a graphical comparison of man-hours required to perform each step in the traditional and formal approaches to RTA requirements capture through test case generation as depicted in Figure 5.

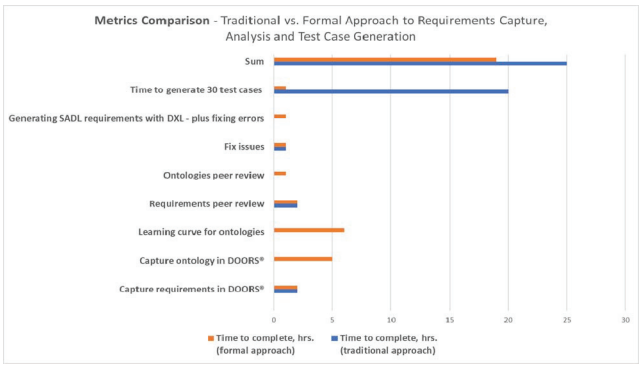

Figure 6

The time in man-hours required to complete all steps up to but not including the generation of test cases (30 test cases were generated in this experiment) were 5 hours for the traditional approach vs. 18 hours for the formal approach. However, the manual generation of requirements-based test cases is the most time-consuming portion of this process. When including the generation of test cases (30 test cases were generated in this example), we saw a 24% reduction in man hours for the full process including all steps shown in Figure 6 when using the formal approach (19 hours for the formal approach vs. 25 hours for the traditional approach). This is discussed in more detail next.

IV. Test Cases Generation Comparison

This section further describes the results of the side-by-side test case generation comparison.

A. *Man-hours Metrics for Comparison of Traditional vs. Formal Test Case Generation*

Manual generation of requirements-based test cases is the most time-consuming portion of this process, requiring 20 hours to manually generate 30 test cases. When using ASSERT™ to auto-generate the 30 test cases, the time required is less than a minute, albeit this is the case when the user is familiar with ASSERT™. When including the generation of 30 requirements-based test cases, the overall man-hours for the formal process is 24% lower than for the traditional manual process (19 hours vs. 25 hours, respectively).

The sample size used for the comparison of Figure 6 was small (7 RTA requirements). However, we can estimate how the comparisons of man-hours for the traditional vs. formal process would scale for large sets of requirements. Figure 7 shows an estimate of man-hours for the traditional vs. formal process applied to a set of 50 requirements. It is important to note that several steps of the process scale 1-to-1 when extrapolating to large requirements sets (capturing requirements in DOORS®, peer reviews and fixing issues). For the formal process steps, there is not a 1-to-1 scaling when extrapolating to larger requirements sets. This is primarily because there is a learning curve involved with the formal process where the time investment goes down as more requirements are captured and analyzed. Therefore, when extrapolating to 50 RTA requirements, the time savings of using the formal process increases to 51% (87 hours to complete the formal process vs. 179 hours to complete the traditional process).

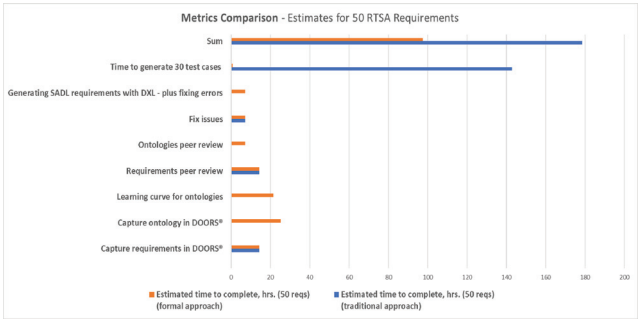

Figure 7

B. *Chaining Test Cases*

One of the key properties of the RTA requirement-base test cases is the chaining of test cases, that is, the ability of a test to pass values to other tests. Test cases are executed according to their defined order in the test plan, which can also be updated in the test tool itself. It is important to correctly order the tests if you want to pass variables from one test case to the other.

Test case RTSA-61	Tests cases RTSA-61A and RTSA-61B
RTSATest Case RTSA-HW-61: Verify status of RTSA is geofenceTrip When fencesStatus is GeoTrip While (fencesStatus of Nfences_p2 is GeoTrip) AND (fencesStatus of Nfences_p3 is GeoTrip)	Test Case RTSA-HW-61A: Verify fencesStatus is GeoTrip When fencesStatusEX of Nfences_p1 is ExcViolated While fencesStatusIN of Nfences_p1 is IncNoViolated
	Test Case RTSA-HW-61B: Verify fencesStatus is GeoTrip When fencesStatusIN of Nfences_p1 is IncViolated While fencesStatusEX of Nfences_p1 is ExcNoViolated

Figure 8

We illustrate chaining with an example. To perform test case RTA-HW-61 we need to know how to evaluate "fencesStatus" equal to "GeoTrip", this information is given by test cases RTA-HW-61A and RTA-HW-61B. This is shown in Figure 8. This chaining is done for each requirement and leads to a big tree structure and exhaustive test, which to do manually is time consuming and error prone.

V. Requirement Implementation Comparison

The previous sections dealt with the comparison of traditional vs. formal capture of RTA requirements and the comparison of manual vs. automated generation of requirements-based test cases. In this section we will describe the use of formal tools developed by NASA to analyze a Simulink model developed to implemment the RTA functions as well as a tool used to convert English requirements to

Linear Temporal Logic (LTL) expressions, from which an automated toolchain developed by Virginia Tech synthesizes safety monitors that can be formally proved to correctly implement the LTL requirements.

A. NASA CoCoSim Tool for Simulink Model Verification

NASA has developed a tool to perform contract based compositional verification of Simulink models, called CoCoSim [19,20]. GE Aviation has implemented the RTA geofence monitor as a Simulink model intended to implement the system requirements. CoCoSim allows users to express safety properties as code segments that implement assume-guarantee reasoning by taking state variables as input and then producing one or more Boolean outputs. CoCoSim compares inputs and outputs of a model to generate a logic True or False answer (i.e. is the Assume/Guarantee contract upheld or not?). If CoCoSim finds that an Assume/Guarantee contract is not upheld, it produces a counterexample showing a set of model inputs and outputs that violate the Assume/Guarantee contract. GE Research and GE Aviation selected a subset of the RTA geofence monitor requirements and developed a set of Assume/Guarantee contracts which were then used to formally verify that the Simulink model implementation of the RTA geofence monitor correctly implements the requirements.

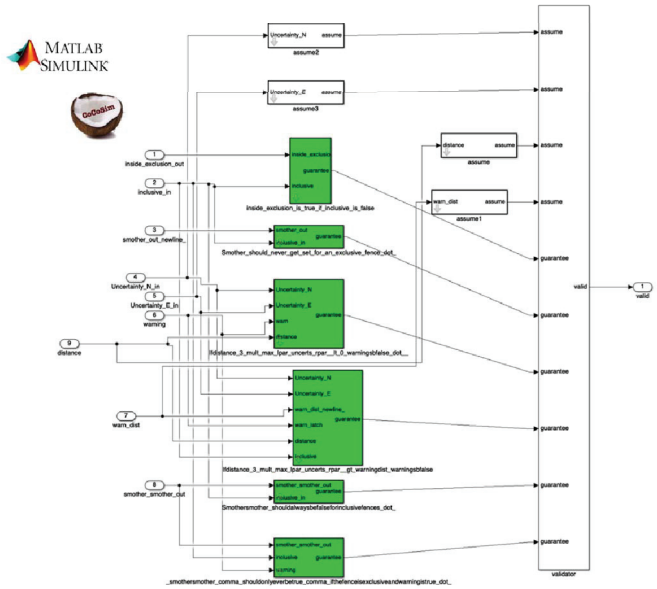

Figure 10

Figure 10 depicts a section of the RTA Simulink model that was verified using CoCoSim. The six shaded Simulink blocks in Figure 10 involve geofence checks and GPS position uncertainty calculations. Assume/Guarantee contracts were formulated from requirements for each of the six blocks. Of those, four were proved to be correctly implemented in the models and CoCoSim generated counterexamples for the other two. The counterexamples help us realize that we did not adequately constrain the size of a geofence and we did not deal adequately with the precision of position and uncertainty calculations at the edge of a geofence. Upon simple corrections to the model parameters CoCoSim then proved all contracts were met and no counterexamples were generated

B. From English Requirements to Linear Time Logic to VHD

NASA has developed a tool called the Formal Requirements Elicitation Tool (FRET) which is used to convert requirements expressed in natural English to Linear Temporal Logic (LTL) expressions [19]. Converting RTA requirements to LTL is the first step in a process to automatically synthesize VHDL-based safety monitors using a toolchain developed by Virginia Tech [18], who is a subcontractor to GE Research on the NASA contract that supports this work. Virgina Tech's toolchain is depicted in Figure 9.

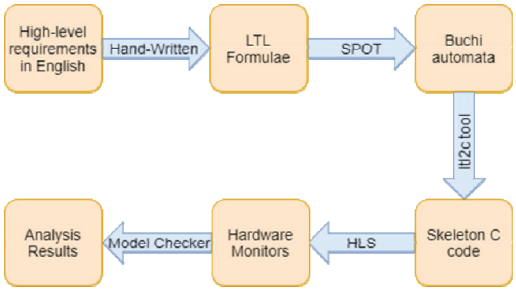

Figure 9

One challenge with the approach depicted in Figure 9 is the difficulty in expressing requirements in Linear Temporal Logic, which is difficult to master and prone to error for non-experts. NASA's FRET tool was developed to overcome this challenge by enabling systems engineers to express requirements in natural English and automating the conversion to LTL expressions. Virginia Tech is using the FRET tool to express English requirements for a complex geofence monitor and auto-generate the LTL expressions that in turn are used as the inputs to the toolchain of Figure 9, resulting in a set of synthesized VHDL monitors that can be formally verified using commercial model checkers. An example English requirement for a simple geofence monitor, along with the LTL expressions generated by FRET are depicted in Figure11.

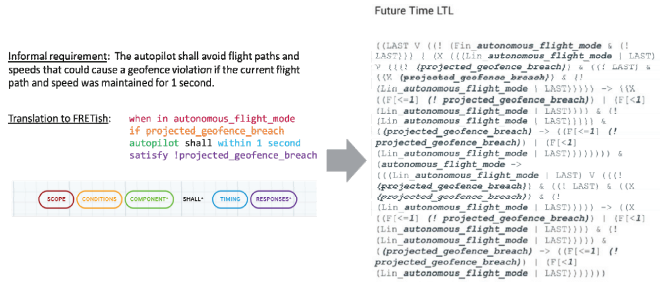

Figure 11

English requirements are entered into the FRET user interface using the "FRETish" structured natural language, and the proper construction of the English requirement in FRETish is aided by prompts to include basic elements of a requirement

such as SCOPE, CONDITIONS, COMPONENT, "SHALL" statement, TIMING and system RESPONSE. The simple English geofence monitor entered into FRET (Figure 11, left) is automatically converted to an LTL expression (Figure 11, right). The complexity of the LTL expression produced illustrates why generating LTL expressions by hand is difficult and error prone. As part of our ongoing study, we will monitor the time it takes to generate LTL expressions for a complex geofence monitor by hand vs. using the FRET tool to auto-generate the LTL expression. Once the LTL expressions are generated, we will synthesize VHDL monitors using the toolchain of Figure 9, and those monitors will be formally verified to be correct to the LTL requirements. This process will be compared to a parallel traditional process in which VHDL monitors are generated by first writing English requirements, generating a Simulink model, synthesizing VHDL from Simulink, and testing the VHDL in a testing environment (e.g. ModelSim) using test cases and test procedures all generated using traditional manual means.

The safety artifacts generated by the V&V processes discussed thus far will all be assembled into a safety assurance case using both traditional means and also using a tool developed by NASA, called AdvoCATE, discussed in the next section.

VI. Assurance Case Generation Comparison

This section describes the initial results of the side-by-side assurance case generation comparison in two parts. The first subsection explains the comparison of capturing the operational risk analysis using the traditional and the formal approach. The second subsection describes our plan to conduct a comparison generating the assurance case manually vs. using the AdvoCATE tool.

An assurance case is a reasoned and compelling argument, supported by a body of evidence, that a system, service or organization will operate as intended for a defined application in a defined environment [17]. The assurance case for the RTA starts with performing an operational risk assessment (ORA) of the system for a specific concept of operations (ConOps) (e.g., BVLOS operations, operations over people). The key hazards associated with a particular ConOps are described and a set of hazard mitigation strategies are developed. From these hazard mitigations strategies, we derive system level requirements. We describe first the ConOps and ORA that are used for studying the comparison analysis.

The UAS system that includes the RTA subsystem shall perform the following ConOps:

CONOP_002: The UAS shall be able to perform a search operation over a geographical area.

Employing operational risk analysis, the risks and the risk mitigation strategies are identified to ensure that no safety hazards will occur during the mission. A subset of these are captured as safety requirements, which are shown below.

UAS_001: An operator of the UAS shall be capable of setting an area of operation that will restrict a flight operation within a geographical area both laterally and vertically.

UAS_009: In an autonomous mode, the system shall have the capability to initiate a FLIGHT_TERMINATION in the event of a detected failure mode.

UAS_008: While the vehicle is in flight, any condition resulting in the vehicle leaving the area of operation, including the following, shall result in FLIGHT_TERMINATION: 1 Commanded Altitude Failures (ORA 1.22, 1.23), 2. Return Home Failures (ORA 1.24, 1.25), 3. Flight Plan Failures (ORA 1.26, 1.27), 4. Commanded Flight Terminate Failures (ORA 1.38).

Now having this baseline, the next 2 subsections explain the initial results and our plan for the comparison of the traditional vs. formal method approach

A. Capturing ORA Analysis Comparison

This subsection explains the process of capturing ORA using Excel and using the AdvoCATE tool. AdvoCATE is a tool set developed by NASA that provides unique automation features to support the development of assurance cases, and methodologies for safety argument development such as GSN and Bowtie method [15,16].

One key aspect of the AdvoCATE tool uses the Bowtie method as a risk evaluation method that can be used to analyze and communicate how high-risk scenarios develop. The essence of the bowtie consists of plausible risk scenarios around a certain hazard and ways in which these risks can be mitigated. The bowtie methodology has the following main goals: Provide a structure to systematically analyze a hazard, help decide whether the current level of control is sufficient and increase risk communication and awareness.

ID	Req. Type	Draft Requirement	Rationale
CONOP_008		The UAS shall be capable of operating up to 3 UAVs simultaneously.	Per SOW
CONOP_009		The UAS shall be able to utilize a secondary predetermined location for landing in case of emergency needs.	Secondary locations will be set up as an alternate landing zone if there was a need to clear airspace for manned traffic.
CONOP_010		The UAS shall be able to change flight plans during flight.	Redirection for new search grids, emergency maneuvers, etc.
UAS_005		While the vehicle is in flight, any of the following conditions resulting in an out of control condition shall result in FLIGHT_TERMINATION: 1. Loss of one or more propellers (ORA 1.2) 2. Failure of structural integrity (ORA 1.4) 3. Loss of thrust from one or more propellers (ORA 1.6) 4. Uncommanded level of thrust from one or more propellers (ORA 1.8) 5. Loss of electrical power to motors (ORA 1.10) 6. Failure to power motors within specification (ORA 1.12, 1.13, 1.14, 1.16, 1.19) 7. Mode Transition Failures (ORA 1.33, 1.34, 1.35, 1.36, 1.37)	While the vehicle is in flight, if any out of control condition is detected, a visible and audible warning signal will be issued the flight will be terminated.

Figure 12

Now we proceed to describe the initial findings of the experiment. Figure 13 shows the bowtie model of the ORA analysis of the RTA captured in the AdvoCATE tool. The benefits are a clear representation of the ORA using the

bowtie methodology and a graphical representation. On the other hand, the traditional approach uses Excel to capture the ConOps and ORA analysis, as shows Figure 12. This approach does not provide bowtie methodology which is a methodology that can help to understand and communicate the ORA.

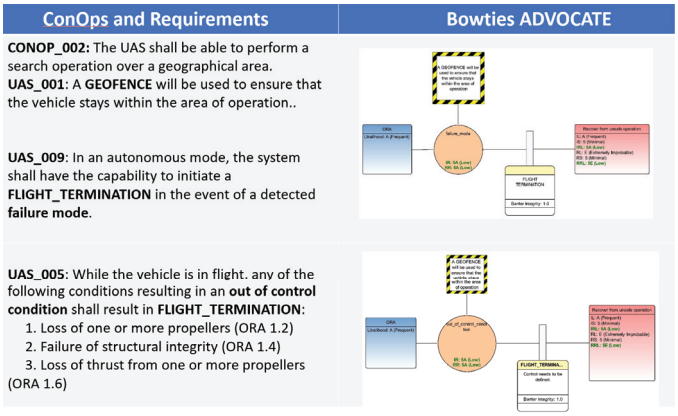

Figure 13

B. Generating Assurance Case From ORA

GSN is a graphical argument notation which can be used to document explicitly the elements and structure of an argument and the argument's relationship to evidence [17]. In GSN, the claims of the argument are documented as goals and the items of evidence are cited in solutions. In order for an assurance case to be developed, discussed and reviewed, it is necessary that the assurance case is clearly documented. By appealing to core concepts of argumentation, GSN helps address this objective.

The experiment in this subsection is to generate GSN from ORA. The AdvoCATE tool automatically generates the GSN from the Bowtie model. Figure 14 shows the GSN generated by the tool. On the other hand, the traditional approach required manual construction of the GSN.

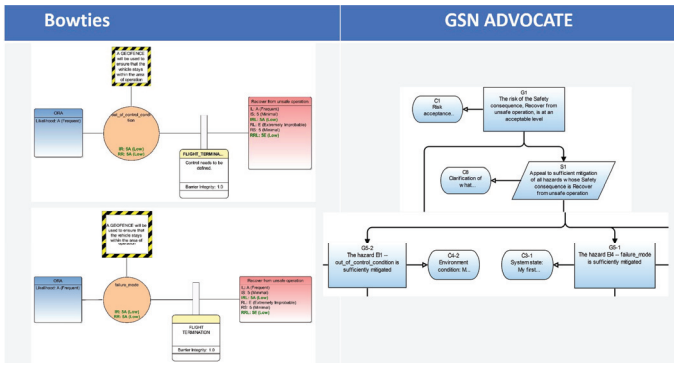

Figure 14

Table 4 shows the comparison, where the main advantage of the AdvoCATE tool is the automatic generation of GSN and the Graphical view, and both tools gives some automation for traceability of ConOps and ORA.

	Comparison criteria		
	Generation of GSN	***Traceability of ConOps and ORA***	***Graphical view***
Excel	Manual	Yes	No
AdvoCATE	Automatic	Yes	Yes

Table 4

We presented a very preliminary comparison of representing fragments of an ORA using Excel and a text-based assurance case vs. a GSN-based assurance case using the AdvoCATE tool. Going forward we will develop a complete assurance case for a UAS operation using the RTA subsystem in AdvoCATE and doing a side-by side comparison to a traditional assurance case. We will establish metrics of comparison that will include quantitative metrics (i.e. comparison of man-hours needed to create an assurance case using the two approaches) as well as qualitative metrics (i.e. feedback from the FAA on the comparative ease of evaluating the two assurance cases).

VII. Conclusion

This paper presented a side-by-side comparison of a complete V&V process for the RTA using both traditional and formal methods-based V&V and showed the benefits of formal tools applied at various early stages of the V&V process. For the requirement capture and test case generation part of the V&V process, a cost metric was presented. A good metric presented was man-hours required to complete requirement capture and test case generation. This metric was rigorously captured to ensure accurate comparisons of cost for the two approaches. For the requirement implementation and assurance case generation a plan and initial results were presented, but currently a generation of man-hours metrics for these steps of the V&V process is under development.

VIII. Acknowledgment

This work is supported by NASA SWS Grant #80NSSC19M0239.

References

[1] K. Siu *et al*., "Flight critical software and systems development using ASSERT™," *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*, St. Petersburg, FL, 2017, pp. 1-10.doi:10.1109/DASC.2017.8102059.

[2] ASTM F3269-17, Standard Practice for Methods to Safely Bound Flight Behavior of Unmanned Aircraft Systems Containing Complex Functions, ASTM International, West Conshohocken, PA, 2017, www.astm.org.

[3] McMillan C, Crapo A, Durling M, Li M, Moitra A, Manolios P, Stephens M, Russell D. Increasing development assurance for system and software development with validation and verification using ASSERT™. SAE Technical Paper; 2019 Mar 19.

[4] K. Siu *et al*., "Flight critical software and systems development using ASSERT™," *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*, St. Petersburg, FL, 2017, pp. 1-10.doi:10.1109/DASC.2017.8102059

[5] GE Global Research, “Semantic Application Design Language (SADL)”, http://sadl.sourceforge.net/

[6] IBM Rational DOORS and Rational DOORS Web Acces. https://www.ibm.com/support/knowledgecenter/en/SSYQBZ_9.6.1/com.ibm.doors.requirements.doc/helpindex_doors.html

[7] A. Crapo, A. Moitra, “Toward a unified English-like representation of semantic models, data, and graph patterns for subject matter experts,” International Journal of Semantic Computing, Vol. 7, No. 3, 2013, pp. 215-236.

[8] A. Crapo, A. Moitra, C. McMillan, D. Russell. “Requirements capture and analysis in ASSERT™,” to appear in IEEE Requirements Engineering Conference (RE17).

[9] Research Triangle Institute, “The economic impacts of inadequate infrastructure for software testing,” NIST Planning Report 02-3, May 2002.

[10] Li, Meng, et al. "Requirements-based Automated Test Generation for Safety Critical Software." *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*. IEEE, 2019.

[11] D. Boren, “Management of test complexity for emerging safety critical control systems program,” Air Force Office of Scientific Research Final Report, May 2006.

[12] J. Offutt, A. Abdurazik, “Generating tests from UML specifications,” 2nd International Conference on Unified Modeling Language, Fort Collins, CO, 1999.

[13] J.Rushby, "Automated test generation and verified software," Working Conference on Verified Software: Theories, Tools, and Experiments, Springer, Berlin Heidelberg, 2005.

[14] Z. Awedikian, "Automatic generation of test input data for MC/DC test coverage," Soccer Lab, Ecole Polytechnique de Montreal.

[15] Denney, Ewen, and Ganesh Pai. "Tool support for assurance case development." *Automated Software Engineering* 25.3 (2018): 435-499.

[16] Denney, Ewen, Ganesh Pai, and Iain Whiteside. "Model-driven development of safety architectures." *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2017.

[17] Kelly, Tim, and Rob Weaver. "The goal structuring notation–a safety argument notation." *Proceedings of the dependable systems and networks 2004 workshop on assurance cases*. Citeseer, 2004.

[18] J. Stamenkovich, L. Maalolan and C. Patterson, "Formal Assurances for Autonomous Systems Without Verifying Application Software," *2019 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED UAS)*, Cranfield, United Kingdom, 2019, pp. 60-69, doi: 10.1109/REDUAS47371.2019.8999690.

[19] Mavridou, Anastasia, et al. *Evaluation of the FRET and CoCoSim tools on the ten Lockheed Martin cyber-physical challenge problems*. Technical report, TM-2019-220374, NASA, 2019.

[20] Bourbouh, Hamza, et al. "CoCoSim, a Code Generation Framework for Control/command Applications: An Overview of CoCoSim for Multi-Periodic Discrete Simulink Models." (2020).