Carnegie Mellon University
Software Engineering Institute

A Model Problem for Assurance Research: An Autonomous Humanitarian Mission Scenario

Gabriel A. Moreno
Anton Hristozov
John Robert
Mark Klein

July 2024

TECHNICAL NOTE
CMU/SEI-2024-TN-001
DOI: 10.1184/R1/26322166

Software Solutions Division

Table of Contents

List of Figures

List of Tables

Acknowledgments

We appreciate the contributions of Stephen Beck, Jason Popowski, David Walbeck, and Dionisio de Niz, who helped create the model problem described in this report.

Abstract

This report describes a model problem to support research in large-scale assurance. In this report, the model problem is illustrated using a scenario that involves an emergency response agency on a humanitarian mission that must be carried out autonomously by an unmanned aerial vehicle to deliver life-saving supplies in a disaster zone. In addition to describing this mission, we describe the architecture of the system used to accomplish the mission and a number of assurance issues that should be addressed. Although the model problem is based on this particular scenario, it represents use in other domains where the same assurance issues can be present. The model problem we present in this report can be used not only to drive the research in this area of assurance but also to demonstrate possible solutions.

1 Introduction

We at the Software Engineering Institute (SEI) are conducting research in the area of large-scale assurance with the goal of reducing the time and effort required to (re-)assure large systems. We consider an *assured system* to be a system for which suitable evidence has been gathered from verification and validation and sufficient arguments have been made to have confidence that the software system is certified for operational use and will work as intended. This notion of system assurance extends beyond security to encompass multiple levels of criticality and a wide range of architecturally significant concerns (e.g., performance, modifiability, safety, reliability).

The increasing scale of systems and their resulting complexity make it difficult to combine capabilities from separately developed systems or subsystems to incorporate innovations and subsequently re-assure systems with speed and confidence. This difficulty is due, in part, to a system's scale, which is not just about how large—by whatever measure is chosen—a system is. Instead, the challenges of scale arise predominantly from the complexity of a system's interactions.

The increasing scale of systems causes unexpected interactions that have not yet been exposed in the context where subsystems have been developed or where the system has been executed. These unexpected interactions can occur due to new contexts, which include new physical and computational environments, interactions with new subsystems, or changes to existing integrated subsystems.

The challenge to assure systems in these circumstances stems from the inability to automatically integrate the complex interacting assurance techniques (e.g., those required for control stability, timing, security, logical correctness) from a system's multiple interacting subsystems. Moreover, the lack of awareness of assurance interdependencies and the lack of effective reuse of prior assurance results leads to considerable re-assurance costs. These costs are due to the need for extensive simulations and tests to discover the interactions among multiple subsystems, especially cyber-physical systems.

Consequently, research is needed to address the challenges to achieving cost-effective (re-)assurance of large systems. To support this research, we present a model problem and scenario that represents it in this report that, while relatively small, reflects the challenges that must be addressed in large-scale assurance. When considering design issues, "a model problem is a reduction of a design issue to its simplest form from which one or more model solutions can be investigated" [Hissam 2004]. The model problem we present in this report poses assurance issues, and it can be used to drive the research for solutions to those assurance issues and demonstrate those solutions.

Our model problem uses a scenario that describes an unmanned aerial vehicle (UAV) that must execute a humanitarian mission autonomously. In this mission, the UAV must fly to a specific location and drop life-saving supplies to people who are stranded and unreachable by land after a natural disaster has altered the terrain and isolated the inhabitants. This scenario, described in Section 2, demonstrates the model problem to research, present, and compare assurance approaches.

We describe the model problem at a high level without any design or implementation details.[1] We avoid prescribing specific forms of analysis or assurance results to be used for system assurance. Some or all of these elements will be needed to demonstrate how assurance approaches address the assurance issues described in Section 5. However, we chose to leave these elements unspecified so that the assurance solutions can focus on addressing the general assurance issues we describe later in the report without limiting which forms of assurance to apply with specific implementation details. The model problem still provides the context for applying assurance research results (e.g., showing how to assure a safety-critical function involving more than one subsystem).

[1] An open source implementation of the model problem, including design or implementation details, is available on GitHub (https://github.com/cmu-sei/FALSA-model-problem) [Moreno 2024].

2 Model Problem Mission Context

In this section, we describe the scenario we use to illustrate the model problem. In this scenario, an emergency response agency acquired UAVs to help people effected by a natural disaster.

2.1 Scenario Context

In recent years, several natural disasters (e.g., hurricanes, earthquakes) have occurred that caused destruction and left people isolated and unreachable, stranded in homes or other buildings. Despite the best efforts by rescue and evacuation operations, the progress was slow, and some people remained unreachable for days. However, some of these people required life-saving supplies right away, and by the time rescuers got to them, in some cases, it was too late.

2.2 Scenario Description

In this scenario, a natural disaster has occurred. The agency in charge of handling emergency response must provide scarce life-saving supplies and deliver them only if certain conditions are met; this approach ensures the supplies are delivered when they are truly needed. More specifically, these supplies must be delivered at specific locations within specified time windows.

The emergency response agency has acquired new UAVs that can deliver the needed supplies autonomously. These UAVs can be invaluable since they can take off, fly to a programmed destination, and drop supplies before returning to the initial launch location.

The UAV vendor affirms that its UAVs can execute these types of missions while meeting the associated stringent requirements. However, there may be unforeseen interactions that the vendor may not have discovered during testing that may occur among the subcontracted parts that were integrated into the UAV. For these reasons, the emergency response agency should require additional assurance from the vendor that the UAVs can execute this mission and its requirements.

2.3 Mission Description

In this section, we provide details about the emergency response agency's mission to provide needed supplies to isolated people affected by a natural disaster. The mission consists of multiple steps, most of which are executed autonomously.

Initially, the UAV operator uses the ground control station (GCS) to enter the drop parameters, which include the coordinates of the drop location, the time window for dropping the life-saving supplies, and an aerial image of the drop site that is used to confirm the supplies are dropped at the right place. These parameters are automatically loaded into the UAV.

Using the GCS, the operator sends authorization to the UAV for takeoff, and the UAV executes the rest of the mission autonomously. The only possible actions the operator might take after providing authorization for takeoff is to abort the mission, which causes the UAV to immediately return to base (RTB).

The autonomous execution of the mission consists of several steps, starting with taking off and flying to the drop location. Once the UAV reaches the drop location, it must check that the site matches the pre-loaded aerial site image and that the current time is within the specified time window for dropping the supplies. Only when these conditions are met should the UAV drop the supplies. Otherwise, it must RTB with the supplies.

At any point during the mission, if the UAV determines that it will not be possible to complete the mission (e.g., strong headwinds cause higher power usage), it must RTB.

After dropping the supplies, the UAV must RTB.

Table 1 describes all the steps of the mission.

Table 1: UAV Mission Steps

Mission Step Label	Mission Step	Responsibility	Description
S-1	Load drop parameters.	Operator	The operator enters the drop parameters into the GCS, and the parameters are loaded into the UAV. Drop parameters include latitude and longitude coordinates, altitude, image of the site where supplies are to be dropped, and the time window for dropping the supplies.
S-2	Authorize takeoff.	Operator	The operator provides authorization for the UAV's takeoff.
S-3	Fly to the drop location.	UAV	The UAV takes off and flies to the drop location.
S-4	(optional) Abort the mission.	Operator	At any time after S-2 but before S-8, the operator can abort the mission from the GCS. When this occurs, the UAV executes an RTB.
S-5	(optional) Abort the mission.	UAV	If the UAV determines that it will be unable to complete the mission (e.g., strong headwinds will deplete the battery before returning to base if mission execution is continued), the UAV executes an RTB.
S-6	When reaching the drop location, compare the site image to the site location.	UAV	If the drop location does not match the loaded site image, the UAV executes an RTB.
S-7	Check the delivery time window.	UAV	If the drop location is reached outside of the delivery time window, the UAV executes an RTB.
S-8	Drop the supplies.	UAV	The UAV drops the supplies at the drop location.
S-9	Return to base.	UAV	The UAV executes an RTB.

3 High-Assurance Functions

There are several functions that must be executed to drop supplies only under specified conditions. Given the importance of this mission requirement, these are considered *high-assurance functions* (HAFs), because it is important that they perform their function only if their preconditions are satisfied and only if they are triggered by corresponding *high-assurance signals* (HASs). (We describe HASs in Section 3.1.)

We list the HAFs for the model problem in Table 2. Even though there are other important functions for the mission (e.g., flying to the drop location), the HAFs ensure that the UAV will not drop the supplies at the wrong place and time, even if the UAV fails to fly to the drop location.

HAFs are classified as *safety critical* if their execution under the proper conditions must be assured. For example, HAF-4 is safety critical because, if the operator aborts the mission, it is critical for the safety of the mission that the abort function is executed. On the other hand, a function such as taking off (HAF-3) is not safety critical because, even if failure to execute prevents the successful completion of the mission, it does not hinder safety.

Table 2: High-Assurance Functions

Function Label	Function	Safety Critical	Description
HAF-1	Set the destination parameters.	No	Unaltered destination parameters (i.e., location, time window) are received by the UAV. This function can be reversed by clearing the destination parameters.
HAF-2	Clear the destination parameters.	Yes	The destination parameters are cleared from the UAV, which prevents a takeoff command from being accepted.
HAF-3	Take off.	No	The UAV takes off and carries out the mission. No further operator input is required other than to abort the mission. This function can be reversed with the abort function.
HAF-4	Abort the mission.	Yes	The mission is aborted, and the UAV must RTB.
HAF-5	Unlock the release mechanism.	No	The mechanism to release the supplies is unlocked. This function can only be executed when the UAV is within a predefined range from the destination and within the destination time window.
HAF-6	Lock the release mechanism.	Yes	The mechanism to release the supplies is locked. This function must be executed if the mission is aborted.
HAF-7	Drop the supplies.	No	The supplies are dropped. This function should only be executed after the destination has been confirmed (e.g., by visual image matching). Supplies will be dropped only if the release mechanism has been unlocked, but this is not simply a logical check; it must have been unlocked using HAF-5 before the drop.

3.1 High-Assurance Signals

A high-assurance signal (HAS) is an action or information transfer that intentionally contributes to the system properly engaging or activating an HAF or disabling an HAF to contribute to the safety of the system.

From an assurance point of view, HASs are as important as HAFs because they should allow the activation of an HAF only in the proper manner, and they should not prevent the activation of an HAF for safety (e.g., lock release system). Therefore, it is important to assure the following about HASs:

- They cannot be issued or propagated without the appropriate preconditions (e.g., the signal for executing the takeoff function must not be issued before setting the destination parameters).
- They are reliably propagated when they are intended to activate a safety-critical HAF.

4 System Architecture

Figure 1 shows a component diagram of the architecture for the UAV system. The two main subsystems are the UAV itself and the GCS. The GCS provides the interface for the operator to do the following:

1. Load the mission parameters into the UAV.
2. Authorize the UAV for takeoff.
3. Abort the mission if needed.

The UAV carries out the rest of the mission autonomously.

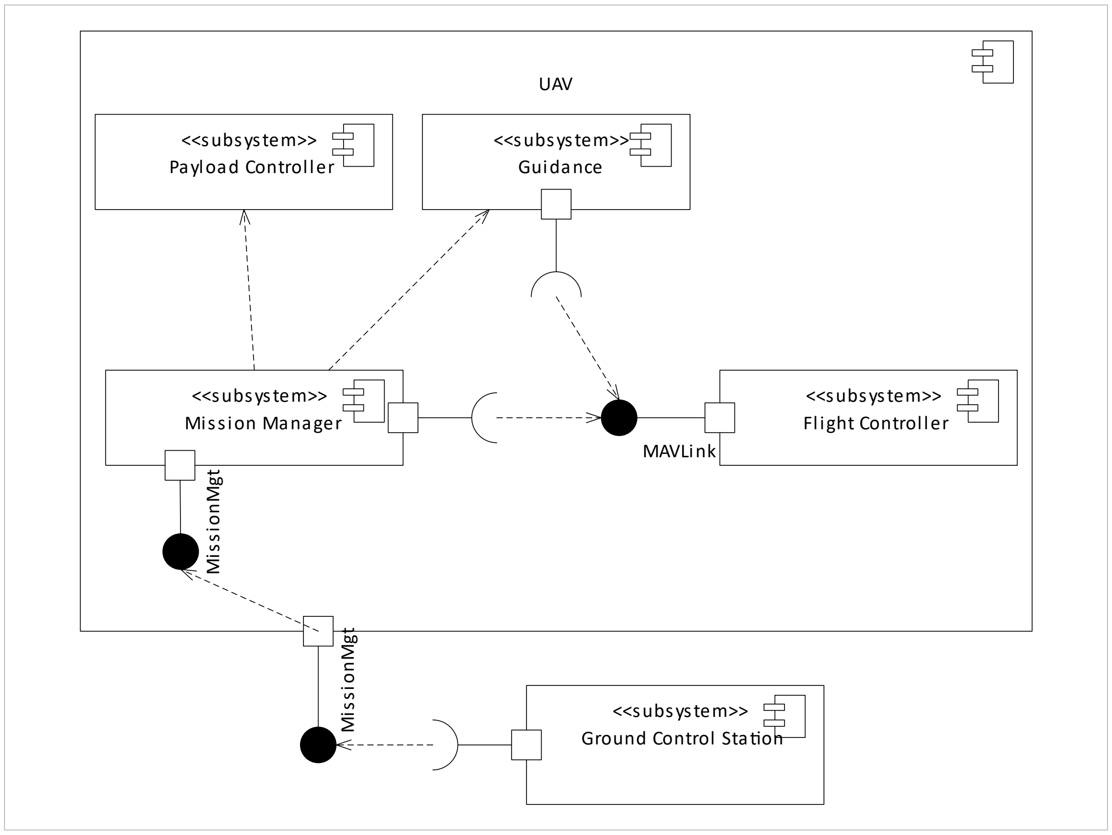
UAV
<<subsystem>> Payload Controller
<<subsystem>> Guidance
<<subsystem>> Mission Manager
<<subsystem>> Flight Controller
MAVLink
MissionMgt
MissionMgt
<<subsystem>> Ground Control Station

Figure 1: System Component Diagram

The main functions included in a UAV's architecture are guidance, navigation, and control (GNC), which allow controlling its movement and enable its accurate navigation [Isser 2021]:

- *Guidance* controls the trajectory the UAV follows to the destination.
- *Navigation* estimates the state of the UAV, including location, velocity vector, and attitude.
- *Control* issues actuation commands to execute the guidance commands while controlling the UAV's stability.

These functions can be organized in different ways, from federated architectures to integration into a single package. In the model problem UAV's architecture, the guidance function is provided by the *Guidance* subsystem, while navigation and control are functions provided by the *Flight Controller* subsystem.

The *Flight Controller* is realized by an autopilot, such as PX4 [PX4 Autopilot 2024] or ArduPilot [ArduPilot 2024], which is configured in a mode that disables its guidance function. All the sensors that the autopilot uses for navigation and control (e.g., magnetometer and inertial measurement unit) are part of this subsystem. The *Flight Controller* provides a *MAVLink* interface [Koubâa 2019] that other subsystems use to interact with the *Flight Controller* to either get information from it (e.g., estate estimate) or send commands to it (e.g., velocity vector for the control function to execute).

The *Payload Controller* subsystem controls the hardware used to drop supplies when commanded. The *Mission Manager* subsystem provides the high-level control of the UAV to execute the mission. After receiving the mission parameters from the GCS, it controls all aspects of the mission, including

- providing *Guidance* with the waypoints to fly to
- monitoring the estimated position provided by the *Flight Controller* to determine when the drop location has been reached
- commanding the *Payload Controller* to drop the supplies if all the necessary conditions are met
- aborting the mission if it is no longer feasible or if it is otherwise commanded to abort

These subsystems are deployed onto different devices as shown in Figure 2. The *Flight Controller* runs on a specialized computer (i.e., the *Flight Computer*), which is suitable for real-time control of the UAV and includes the sensors needed for navigation and control. The rest of the UAV's subsystems execute on the *Mission Computer* (a separate dedicated computer). The *Flight Computer* and *Mission Computer* are connected with a high-speed dedicated wired connection. The *Ground Computer*, where the GCS software executes, is connected via a wireless connection to the UAV. However, that connection is assumed to be secure and reliable, and thus is not a concern for the assurance of the system.

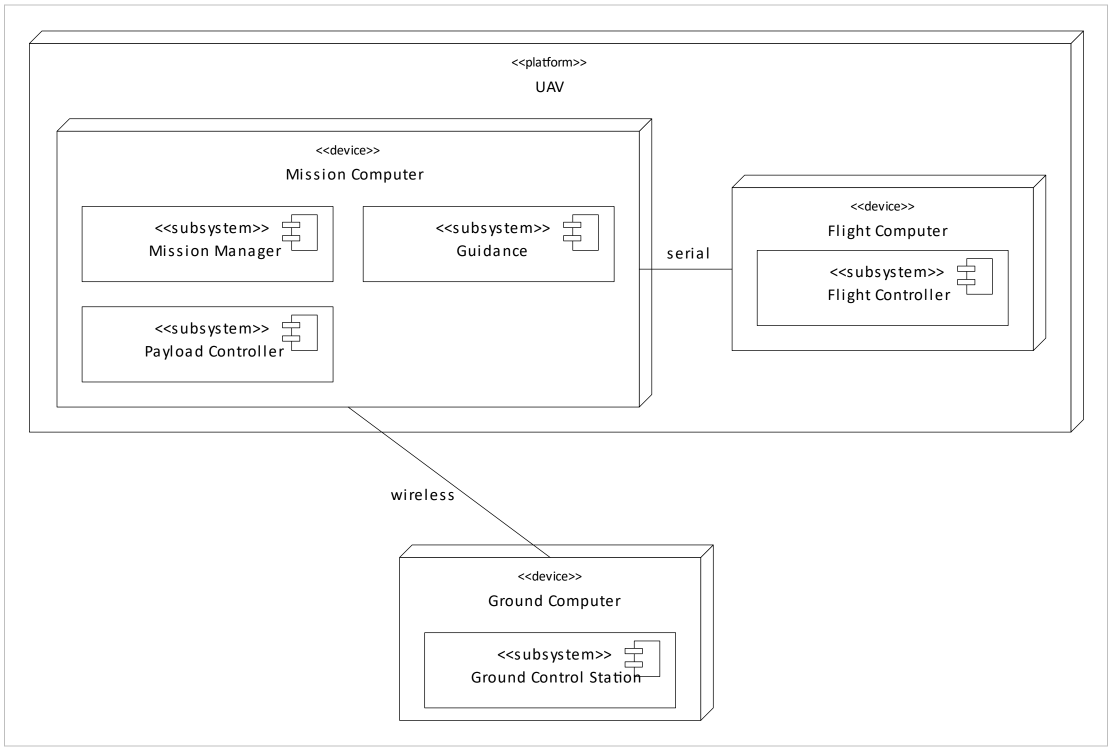

Figure 2: System Deployment Diagram

4.1 System Behavior Model

Considering the importance of system behavior in this model problem (especially the behavior as it relates to HAFs and HASs), it is of utmost importance to assure the properties related to system behavior. For example, in the scenario described in Section 2, an analyst will want to assure that the UAV cannot take off without the mission parameters first being set. Different forms of analysis can be used to ensure that happens, including model checking a formal model of the behavior and simulating a state machine modeling the behavior [Liu 2013].

The unified modeling language (UML) state machine shown in Figure 3 models the behavior of the UAV system in greater detail than described in previous sections. This increased specification detail enables analyses that can be used to assure system behavior.

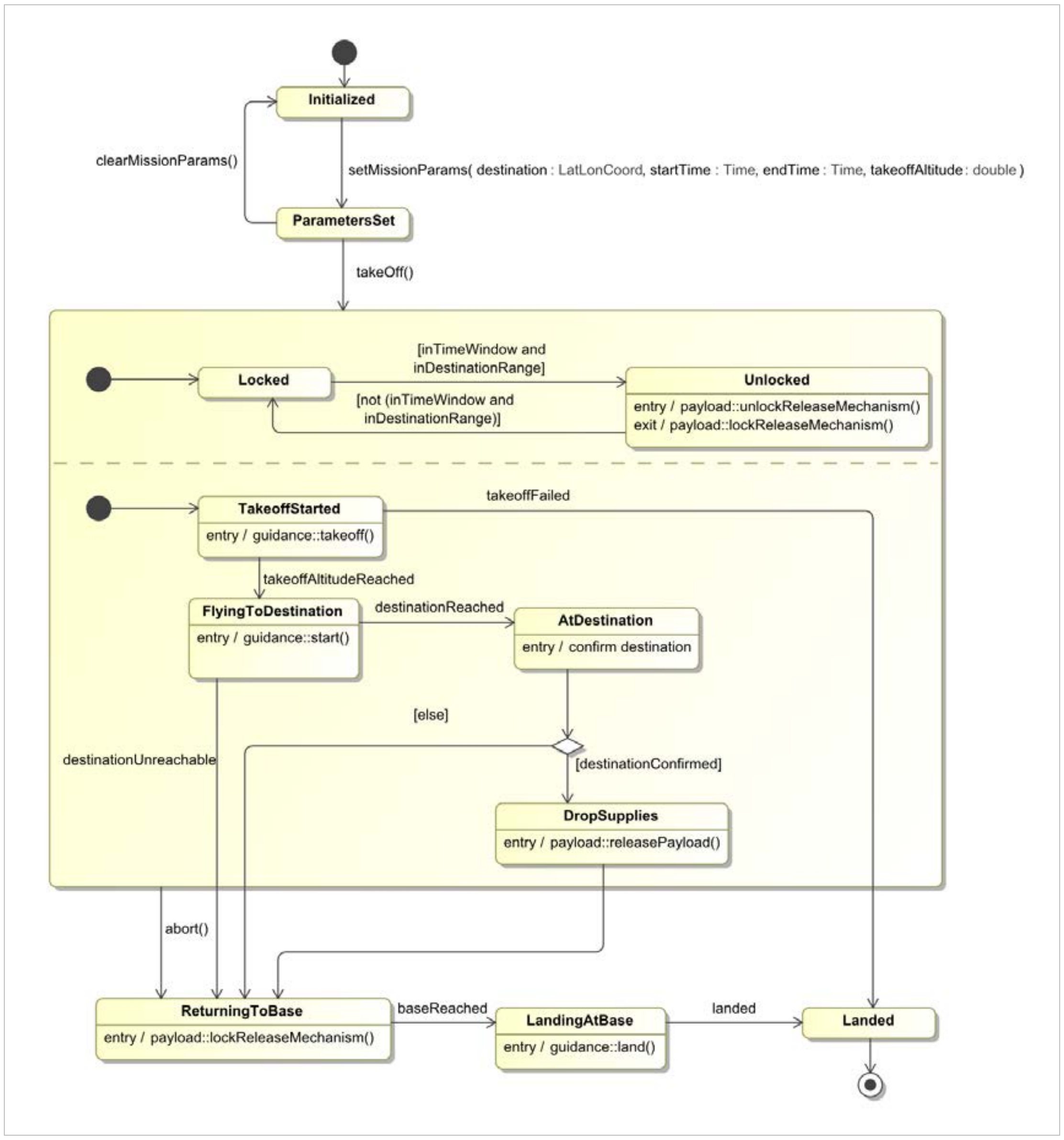
Initialized
clearMissionParams()
setMissionParams(destination : LatLonCoord, startTime : Time, endTime : Time, takeoffAltitude : double)
ParametersSet
takeOff()
[inTimeWindow and inDestinationRange]
Locked
Unlocked
[not (inTimeWindow and inDestinationRange)]
entry / payload::unlockReleaseMechanism()
exit / payload::lockReleaseMechanism()
TakeoffStarted
entry / guidance::takeoff()
takeoffFailed
takeoffAltitudeReached
FlyingToDestination
entry / guidance::start()
destinationReached
AtDestination
entry / confirm destination
[else]
destinationUnreachable
[destinationConfirmed]
DropSupplies
entry / payload::releasePayload()
abort()
ReturningToBase
entry / payload::lockReleaseMechanism()
baseReached
LandingAtBase
entry / guidance::land()
landed
Landed

Figure 3: State Machine Diagram for UAV System Behavior

5 Assurance Issues

The goal of the model problem is to provide the context to research and develop assurance approaches that can address different assurance issues that are key to achieving large-scale assurance and reducing assurance effort and cost.

The following are these key assurance issues:

- Different kinds of assurance analyses and results (e.g., response time analysis, temporal logic verification, test results) are needed and must be combined into a single assurance argument.
- Each analysis makes different assumptions, which must be consistent across analyses and must be satisfied.
- Different subsystems can be developed by different organizations, which provide assurance results for the subsystem.
- The different assurance analyses and results used in the assurance argument may have different levels of strength—from the simple testing of a few cases to exhaustive model checking. Therefore, conclusions about claims supported by the assurance argument must consider these different strength levels.
- It may not be feasible or desirable to build a complete assurance argument before some system assurance results can be provided. Therefore, it should be possible to build the assurance argument incrementally.
- The system is likely to evolve due to changes or upgrades in individual subsystems. It should be possible to reuse assurance results when only part of the system changes.

The following section provides some claims for assurance that researchers can use to demonstrate how advances in the state of the art and practice can tackle these issues.

5.1 Claims for Assurance

An assurance argument proves or supports a claim about a system (e.g., the system is safe to operate in a given environment). Different stakeholders have different assurance concerns about the UAV:

- its ability to successfully execute the mission
- its safety
- its security so that a malicious actor cannot force it to deviate from its intended mission

The model problem enables researchers to explore the assurance concerns of different stakeholders. Table 3 lists the top-level claims that can be used to drive different assurance efforts. In general, a claim is typically decomposed into other claims, as necessary, to prove or support the claim that the system satisfies the property stated in the top-level claim. Furthermore, proving that the system satisfies a claim can involve analyses in different domains.

For example, in the UAV scenario described in Section 2, proving the last claim in the table requires showing that the control algorithm in *Guidance* can command the UAV to fly the trajectory, but the control algorithm assumes that it has up-to-date state information, which is provided by the *Flight Controller*. Therefore, proving this claim requires showing that the *Flight Controller* provides that information with the needed frequency, and that the transmission of that information to the *Guidance* subsystem does not introduce a delay that results in the information being stale by the time the control algorithm uses it.

Table 3 provides examples of claims that present different assurance challenges. However, the actual approach to decompose these claims, along with the analyses and assurance results used, are determined by the approaches developed to address the assurance challenges inherent in this model problem.

Table 3: Claims for Assurance

	Claim
1	The system will meet the mission requirements.
2	If there is a failure in a non-critical part of the system, the system will be resilient and continue to perform the mission as well as possible.
3	A *safety-critical* HAF is always executed when its HAS is issued.
4	An HAF can never execute without the required preconditions and its HAS.
5	If there is a failure in the execution of an HAF or the propagation of its HAS, the system will always fail safely.
6	A malicious actor cannot make the system execute HAFs and bypass the necessary preconditions.
7	The UAV follows the intended trajectory within its performance thresholds.

6 Conclusion

In this report, we presented a model problem using a scenario that revolved around the successful execution of an autonomous humanitarian mission to deliver life-saving supplies with a UAV in a disaster zone. We described a scenario and the architecture of the system used to carry out the scenario mission, including multiple subsystems and computing platforms. This system and its subsystems represent the assurance challenges experienced in large-scale assurance.

In this report, we also provided a list of assurance issues that solutions should address and a list of assurance claims that cover the concerns of different stakeholders. These assurance claims can be used to demonstrate solutions to the assurance issues posed by the model problem.

References

URLs are valid as of the publication date of this report.

[ArduPilot 2024]
ArduPilot. ArduPilot: Versatile, Trusted, Open. *ArduPilot Website*. July 15, 2024 [accessed]. https://ardupilot.org/

[Hissam 2004]
Hissam, Scott A. & Klein, Mark. *A Model Problem for an Open Robotics Controller*. CMU/SEI-2004-TN-030. Software Engineering Institute, Carnegie Mellon University. 2004. https://insights.sei.cmu.edu/library/a-model-problem-for-an-open-robotics-controller/

[Isser 2021]
Isser, Abraham. *Introduction to Guidance, Navigation, and Control (GNC)*. DSIAC-BCO-2021-172. Defense Systems Information Analysis Center (DSIAC). 2021. https://dsiac.org/wp-content/uploads/2022/10/AD1182620.pdf

[Koubâa 2019]
Koubâa, Anis; Allouch, Azza; Alajlan, Maram; Javed, Yasir; Belghith, Abdelfettah; & Khalgui, Mohamed. Micro Air Vehicle Link (MAVlink) in a Nutshell: A Survey. *IEEE Access*. Volume 7. Pages 87658–87680. 2019. http://doi.org/10.1109/ACCESS.2019.2924410

[Liu 2013]
Liu, Shuang; Liu, Yang; Sun, Jun; Zheng, Manchun; Wadhwa, Bimlesh; & Dong, Jin Song. USMMC: A Self-Contained Model Checker for UML State Machines. Pages 623–626. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. August 2013. https://doi.org/10.1145/2491411.2494595

[Moreno 2024]
Moreno, Gabriel A. FALSA Model Problem. *GitHub*. May 2024. https://github.com/cmu-sei/FALSA-model-problem

[PX4 Autopilot 2024]
PX4 Autopilot. Open Source Autopilot for Drone Developers. *PX4 Website*. July 15, 2024 [accessed]. https://px4.io/

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.			
1. **AGENCY USE ONLY** (Leave Blank)	2. **REPORT DATE** July 2024		3. **REPORT TYPE AND DATES COVERED** Final
4. **TITLE AND SUBTITLE** A Model Problem for Assurance Research: An Autonomous Humanitarian Mission Scenario			5. **FUNDING NUMBERS** FA8702-15-D-0002
6. **AUTHOR(S)** Gabriel A. Moreno, Anton Hristozov, John Robert, & Mark Klein			
7. **PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213			8. **PERFORMING ORGANIZATION REPORT NUMBER** CMU/SEI-2024-TN-001
9. **SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)** SEI Administrative Agent AFLCMC/AZS 5 Eglin Street Hanscom AFB, MA 01731-2100			10. **SPONSORING/MONITORING AGENCY REPORT NUMBER** n/a
11. **SUPPLEMENTARY NOTES**			
12A **DISTRIBUTION/AVAILABILITY STATEMENT** Unclassified/Unlimited, DTIC, NTIS			12B **DISTRIBUTION CODE**
13. **ABSTRACT (MAXIMUM 200 WORDS)** This report describes a model problem to support research in large-scale assurance. In this report, the model problem is illustrated using a scenario that involves an emergency response agency on a humanitarian mission that must be carried out autonomously by an unmanned aerial vehicle to deliver life-saving supplies in a disaster zone. In addition to describing this mission, we describe the architecture of the system used to accomplish the mission and a number of assurance issues that should be addressed. Although the model problem is based on this particular scenario, it represents use in other domains where the same assurance issues can be present. The model problem we present in this report can be used not only to drive the research in this area of assurance but also to demonstrate possible solutions.			
14. **SUBJECT TERMS** large-scale assurance, model problem, assured system			15. **NUMBER OF PAGES** 21
16. **PRICE CODE**			
17. **SECURITY CLASSIFICATION OF REPORT** Unclassified	18. **SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	19. **SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	20. **LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18 298-102