Subject Areas:
Robotics, Software, Systems theory

Keywords:
Autonomous systems, Hazard assessment, Assurance, Requirement formalization, NLP, Resilience simulation

Author for correspondence:
Anastasia Mavridou
e-mail:
anastasia.mavridou@nasa.gov

Towards a Hazard Assessment Workflow for Assurance of Increasingly Autonomous Systems

Lukman Irshad[1,2], Anastasia Mavridou[1,2], Seydou Mbaye[1], Daniel Hulse[1], Irfan Sljivo[1,2], and Hannah Walsh[1]

[1]NASA Ames Research Center, Moffett Field, CA 94035
[2]KBR, Moffett Field, CA 94035

The advent of increasingly autonomous systems poses a significant challenge for design assurance. As human operators are increasingly replaced by automated decision-making, the operational environment must be taken into account at design time. As a result, there is an increasingly large and complex input space to consider during design, and assurance approaches must be adapted to accommodate this paradigm shift. Existing safety assessment methodologies were typically developed for systems under continuous human supervision, and new approaches are needed to assure autonomous systems. In this article, we propose a hazard assessment methodology for the safety assurance of increasingly autonomous systems. The workflow combines Natural Language Processing (NLP)-enabled elicitation of hazards from historical incident data, functional modeling, and scenario-based resilience simulation to augment traditional expert-driven hazard analysis and improve coverage of the hazard space. Requirements derived from this analysis are subsequently formalized in a structured natural language, enabling automatic generation of logical formulæ, which can be analyzed for correctness. The framework is realized through two NASA-developed toolsets: the Hazard Analysis DESigner (HADES) and the Formal Requirements Elicitation Tool (FRET). The methodology is demonstrated on TROUPE, a NASA experimental rover swarm designed to autonomously and cooperatively map unknown terrain.

1. Introduction

Recent advances in artificial intelligence (AI) and machine learning (ML) have paved the way for engineered systems that act with greater levels of autonomy. Experts argue that various routine and specialized operations can and should be offloaded onto machines, especially in environments that are hard to access or can be detrimental to humans [1–3]. This has driven an uptick in the development and deployment of autonomous systems in areas such as disaster management [4] and space exploration [5].

For safety-critical systems, the assurance of safety is paramount and must inherently be integrated into the design process. Given that autonomous systems are expected to operate with minimal human interaction, they must be designed with the inherent resilience to mitigate all known, and even unknown, hazardous conditions as they arise in operations. However, many existing approaches traditionally used for safety assessment have limitations which render them inadequate for future, highly AI-reliant systems [6,7].

Most standards governing the development of airborne systems were developed for systems that are much less reliant on software compared to autonomous systems. Such standards include ARP4761A which provides guidelines for the safety assessment of civil aircraft systems [8,9]. DO-178C governs airborne software consideration through requirements-based testing [10], and ARP4754A provides guidance for the development of highly integrated airborne systems [11]. While these approaches are appropriate for assuring the current physical and software aspects of aircraft, they do not cover the assurance of onboard autonomy. They were created to assure predictable, deterministic software in which a human pilot is assumed to have control of (and thus responsibility for) the operation of the aircraft. As such, they do not account for the dynamic, adaptive behavior of autonomous systems. Moreover, they do not offer guidance on validating AI/ML algorithms and thus cannot be used to construct the risk-informed safety cases that the rigorous assurance of these algorithms would demand.

NASA defines a risk-informed safety case as one that not only demonstrates a system is adequately safe, but that plausible, knowable scenarios leading to system failure have been identified, analyzed, and adequately managed [12]. One of the biggest challenges in building such a risk-informed safety case for autonomous systems is requirements engineering [13]. Safety requirements derived from hazard analysis [14] must be unambiguous, traceable, and amenable to formal verification [13], properties that are difficult to achieve with natural language [15]. While these challenges have been documented in prior work [7,16], bridging the gap between high-level safety intent and formally verifiable specifications remains a critical challenge in the assurance of autonomous systems.

In this article, we propose a hazard assessment workflow that addresses these challenges through the principled integration of tools into the safety assurance process. The framework combines the elicitation of historic knowledge using Natural Language Processing (NLP) and the simulation of the systems resilience to hazards to inform and extend the traditional expert-driven hazard assessment, thereby improving the coverage and understanding of the space of hazardous conditions and scenarios the system may operate under. Requirements derived from this analysis are subsequently formalized using a structured natural language, enabling the automatic generation of mathematical formulae and runtime monitors, with traceability maintained throughout from initial hazard identification to monitor generation. The framework is realized through two NASA-developed toolsets: the Hazard Analysis DESigner (HADES), which provides functional modeling, historical data mining, and hazard simulation capabilities, and the Formal Requirements Elicitation Tool (FRET), which supports requirements authoring, formalization, and analysis. The methodology is demonstrated on TROUPE [17], a NASA-developed experimental rover swarm designed to autonomously map unknown terrain.

This paper makes the following contributions:

- **An eight-step hazard assessment workflow** that connects NLP-based knowledge elicitation, functional modeling and resilience simulation, and formal requirements specification and analysis within a single process. While prior work has applied these tools separately, the contribution lies in their integration and the information flows between them that unlock capabilities unavailable to either tool in isolation. Specifically, this integration enables the iterative, adaptive safety assurance paradigm needed for autonomous systems, where safety is the result of an evolving interplay between the system, its software, and its operating environment.
- **A case study on simulation-grounded requirements elicitation**. Simulation results ground specific requirement thresholds, which are then formalized in FRET, verified for consistency, and automatically transformed into temporal logic formulæthat can be used for runtime monitoring and analysis purposes.

2. Background

In this section, we describe the underlying tools of our proposed hazard assessment workflow.

(a) Hazard Analysis DESigner (HADES)

Hazard Analysis DESigner (HADES) is an envisioned set of capabilities that embody the concept of Computational Functional Hazard Assessment (CFHA). In the CFHA paradigm, the traditional hazard analysis process is supported with and complemented by computational functionality, including data retrieval and analysis, model-based engineering, and simulation, to achieve more complete, traceable, and adaptable hazard assessments [18,19]. Two already-developed components of HADES include the Manager for Intelligent Knowledge Access (MIKA)[1] and Fault Model Design Tools (fmdtools)[2].

MIKA is a Natural Language Processing tool that aids with risk-related knowledge discovery from text corpora with a goal of supporting early design failure/hazard assessments. Specifically, it supports topic modeling using multiple applicable algorithms such as BERTopic and Latent Dirichlet Allocation and derivatives [20], and information retrieval tasks using semantic search powered by sentence-BERT [21]. MIKA has been used to extract risk- and human error-related information from historical incident reports [22,23]. It has further been used to support the development of fishbone diagrams [24], functional hazard assessments [18], and model-based failure modes and effects analyses [25] using the extracted information.

The fmdtools library is a simulation-based framework that can be used to evaluate a system's response to hazardous scenarios to understand and quantify systems resilience [26] and safety. This library can be applied at both the component or system level (e.g., evaluating fault propagation through a system architecture) as well as the agent or system-of-systems level (e.g., evaluating how well operators can maintain safety in a shared environment), and provides specialized capabilities for representing human/component interactions [27], performing risk-informed trades studies [28], and discovering new worst-case hazardous modes [29]. The fmdtools library further includes the Functional Reasoning Design Language[3] (FRDL) specification for representing behavioral interactions to support hazard analysis [30].

Together, this suite of tools can be used to support and complement the hazard assessment process in different ways, by helping identify hazards and their features (MIKA), making it easier to trace the effects of hazard through a system architecture (FRDL) and enabling the statistical simulation of these hazards in a model of system behaviour (fmdtools). These tools are further proposed to better represent the safety characteristics of autonomous systems–where dynamical system interactions between human operators/users and onboard autonomy, the

[1] https://github.com/nasa/mika
[2] https://github.com/nasa/fmdtools
[3] https://nasa.github.io/fmdtools/docs-source/frdl.html

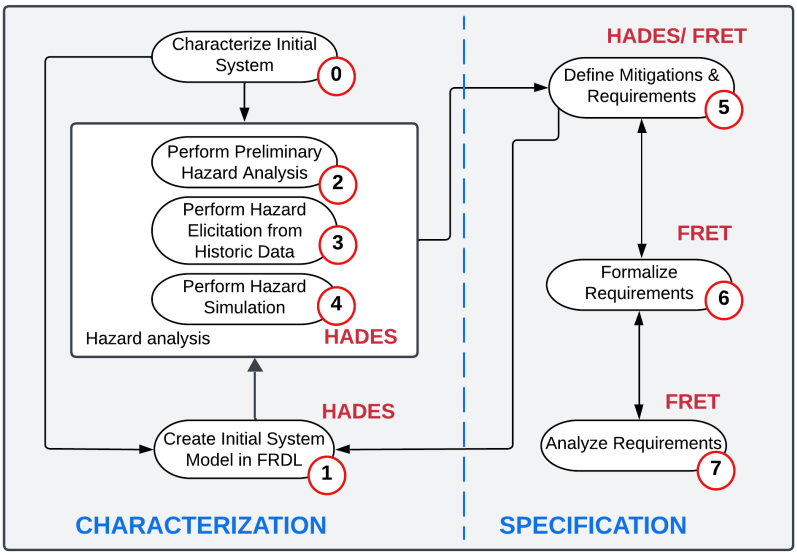


Figure 1: Eight-step hazard assessment workflow integrating system characterization (Steps 0–1), hazard analysis (Steps 2–4), and requirements formalization and analysis (Steps 5–7)

technical functions of the vehicle/aircraft or system, and the external operating environment (1) is expected to change as a part of the autonomy concept and (2) have major implications for safety.

(b) Formal Requirements Elicitation Tool (FRET)

FRET [31] is an open-source tool [32] for writing, understanding, formalizing, and analyzing requirements. In practice, requirements are typically written in natural language, which is ambiguous and, consequently, not amenable to formal analysis. Since formal, mathematical notations are unintuitive, requirements in FRET are entered in a restricted natural language named FRETISH [33] with precise unambiguous meaning. FRET helps users write FRETISH requirements both by providing grammar information and examples during editing, but also through English and diagrammatic explanations to clarify subtle semantic issues. For each FRETISH requirement, FRET generates formulae in a variety of formalisms, including Linear Temporal Logic (LTL) and Lustre [34] code. Moreover, FRET supports interactive simulation of produced formalizations to ensure that they capture user intentions. Through its analysis portal, FRET connects to analysis tools (e.g., [35,36]) by facilitating the mapping between requirements and models/code, and by generating verification code. FRET also supports the realizability analysis of requirements, which can be used to identify conflicting requirements [37,38]. Finally FRET supports test-case generation from requirements with an associated requirements-based coverage metric [39].

3. Methodology

The objective of this work is *to study the integration of hazard analysis and requirements formal specification and analysis, as applied to the TROUPE swarm of autonomous rovers using a tool palette that includes three NASA Ames Research Center tools:* FRET, *fmdtools, and MIKA (the latter two are jointly called HADES).*

To this end, we present an eight-step methodology for the hazard assessment and requirements formalization and analysis of autonomous systems. The steps are shown in Figure 1 and are organized into three phases. The first phase, *System Characterization and Modeling* (Steps 0–1),

establishes the system description and its functional model, which serve as the foundation for all subsequent analysis. The second phase, *Hazard Analysis* (Steps 2–4), identifies and refines hazards through a combination of expert-driven assessment, historical data mining, and resilience simulation. The third phase, *Requirements Engineering* (Steps 5–7), derives, formalizes, and analyzes requirements that mitigate the identified hazards. The steps are:

- **Step 0:** Characterize initial system.
- **Step 1:** Create initial system model in Functional Reasoning Design Language (FRDL).
- **Step 2:** Perform preliminary hazard analysis.
- **Step 3:** Perform hazard elicitation from historic data.
- **Step 4:** Perform hazard simulation.
- **Step 5:** Define mitigations and requirements.
- **Step 6:** Formalize requirements.
- **Step 7:** Analyze requirements.

Although the steps are presented in chronological order, the methodology is inherently iterative. Any step may be revisited during the design lifecycle—for instance, realizability conflicts identified in Step 7 may prompt refinements to mitigations in Step 5 or even revisions to the hazard analysis in Steps 2–4. When a step is revisited, all subsequent steps must also be reconsidered to maintain consistency and traceability across the workflow.

Over the years, we have worked with a variety of formal approaches for the assurance of safety-critical systems. The goal of this study is to explore how such approaches can work together and be integrated within the development process of an autonomous system. To this aim, we use TROUPE–a knowledge transfer project in the Robust Software Engineering group at NASA Ames Research Center–to demonstrate the proposed methodology [17]. The project aims at developing a micro-rover swarm that can autonomously map unknown terrain while using advanced assurance techniques [17,40]. Each rover runs an identical core Flight System (cFS) [41] application. The rovers work cooperatively, each collecting data for different parts of the environment to achieve the overall mission objective. Rovers present challenges that are representative of autonomous systems more broadly: they operate in unstructured, partially unknown environments with minimal human oversight, rely on onboard sensing and AI-driven decision-making to handle situations that cannot be fully anticipated at design time, and must satisfy stringent safety despite the uncertainty. Additionally, TROUPE follows NASA mission development guidelines, providing realistic engineering context for the case study.

In the following subsections, we describe our proposed methodology in detail.

(a) Initial System Characterization

System characterization is the process where system functions, needs, objectives, and operational processes are identified and defined at a high level from the perspective of users. Often, in system design, this is captured through the concept of operations (ConOps).

In the TROUPE case study, each rover can communicate with the ground station, where mission commands (e.g., destination, forced stop, remote navigation) and map data are exchanged. The mapping mission begins when the rover receives a destination command. Each rover is equipped with its own energy storage system, which can be recharged at a base station. The rovers are outfitted with onboard cameras and LIDAR systems that provide 360-degree coverage within a five-meter range. They use these sensors to determine their location and orientation for localization, which in turn supports navigation. When localization data is unavailable, the rover estimates its current pose and location using navigation data. Rovers are expected to avoid obstacles and navigate only through traversable terrain. As the rover moves, it uses camera and lidar data to map the surrounding terrain. This mapping data is transmitted to the ground station at predefined intervals. The rover must always maintain sufficient energy reserves to return to base if necessary. If the destination lies in an area that is not navigable, the

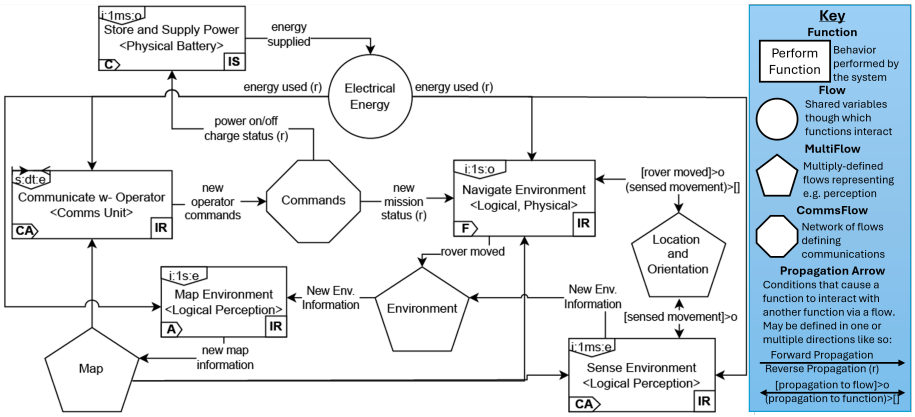


Figure 2: Functional model of a TROUPE Rover modeled using FRDL

rover will abort the mission and return to base. Rovers may also communicate with each other and share mapping data.

(b) System Modeling

As shown in Figure 1, once system characterization is complete, a more detailed system model must be developed to lay the groundwork for the subsequent steps. It is important to note that not all system details may be available at this stage. Therefore, the objective is to model the system to the extent possible using the available data and refine the model as additional information becomes available. For the purposes of this study, we focus solely on an individual TROUPE rover for both modeling and subsequent analysis. While a comprehensive assessment would require a system-of-systems approach including multiple rovers, an energy supply base, and a ground station, such considerations are beyond the scope of this study. The primary goal here is to demonstrate the methodology outlined in Figure 1.

In this methodology, we use the Functional Reasoning Design Language (FRDL) [30] within HADES for system modeling. FRDL was developed to enable more effective representation of system behaviors, interactions, and hazard propagation during early design stages. Its goal is to enable safety assessment practitioners to more accurately analyze the effects of hazardous conditions in their systems, while providing a modeling ontology that is amenable to simulation. FRDL employs a bipartite graph structure, where block-type nodes in the graph represent behavioral elements (such as functions, components, or actions depending on the modeled architecture type), and circle-type nodes represent flows. Flows in FRDL represent shared variables between these entities. The edges in the graph denote relationships such as connections, propagations, or activations. Past work has demonstrated the utility of FRDL in capturing hazards and their propagation more effectively than traditional failure or fault assessment approaches [30].

In this study, we develop a functional architecture, FRDL's defined diagram for supporting the functional hazard assessment process, of the TROUPE rover. The resulting function architecture is shown in Figure 2, where *Store and Supply Power*, *Communicate with Operator*, *Map Environment*, *Navigate Environment*, and *Sense Environment* are functions of the rover, and *Electrical Energy*, *Commands*, *Map*, *Environment*, and *Location and Orientation* are flows (i.e., variables shared between functions). The rover defaults to a standby mode in which minimal *Electrical Energy* is used to keep the *Communicate with Operator* function active, allowing it to receive *Commands* from the ground station. When a destination command is received, the *Supply and Store Power* function provides Electrical Energy to all components associated with each function, thereby activating them. The *Map* function perceives the *Environment* within its vision coverage area and generates

a *Map* of the surroundings, identifying obstacles and non-traversable regions. This *Map* is then passed to the *Sense Environment* function, which uses it along with sensor data gathered from the *Environment* for localization. The resulting *Location and Orientation*, along with the *Map*, are then provided to the *Navigate Environment* function. This function uses the data to identify waypoints, determine the trajectory toward the next waypoint and then physically move the rover's location and guide its orientation along these waypoints by translating energy into locomotion. Localization computations within the *Sense Environment* function can sometimes be delayed, meaning location and orientation data may not always be immediately available. In such cases, the *Navigate Environment* function estimates the rover's location and orientation, updating these estimates once accurate data is received from the *Sense Environment* function. The *Supply and Store Power* function returns the *Electrical Energy* supply to standby mode once the mission is completed (i.e., the destination is reached) or aborted (i.e., the destination is deemed untraversable). Note that while in a real-world scenario the rover would return to base under these conditions, for simplicity the scope of this demonstration focuses solely on reaching the destination. Additionally the following assumptions are made for the rest of the analysis:

(i) The environment in which the rover operates will always contain obstacles.
(ii) The obstacles are static.
(iii) The rovers have built-in safeguards to ensure they always traverse a path only when sufficient power is available.

(c) Hazard Analysis

Early design hazard analysis for a given system involves the identification of potential hazards, their effects, and their severity. These analyses serve two main purposes. First, they support design decisions by helping engineers define mitigations and elicit safety-related requirements. Second, they justify and inform the determination of required design assurance levels for future verification and validation activities. One of the most important processes for identifying these hazards in the design of aircraft is the Functional Hazard Assessment (FHA). FHA [8,42] is an expert-driven process in which analysts identify the high-level functional failures and their effects and severity levels in each relevant phase of system operation. This information is then used to derive appropriate mitigations and requirements, as well as to determine the necessary design assurance levels.

To augment human expertise in the hazard analysis process and better address autonomous systems safety challenges, the authors previously developed the idea of CFHA [18], an approach that relies on the capabilities of software to better identify, explore, and analyze hazardous scenarios where expert knowledge may be limited. In this research, we apply CFHA as implemented in the HADES framework, utilizing tools such as FRDL, MIKA, and fmdtools. The CFHA process within HADES begins with a Preliminary Hazard Analysis, which is subsequently refined and validated using historical data and simulation-based hazard and resilience analyses.

(i) Preliminary Hazard Analysis

The Preliminary Hazard Analysis in CFHA begins with a traditional Functional Hazard Assessment (FHA), where experts populate a hazard table without computational assistance. However, unlike the conventional approach, CFHA leverages FRDL models to support the analysis. These models serve as a foundation for tracing hazard propagation paths, enabling experts to identify hazard effects as well as their underlying mechanisms or causes. Guidance on how to trace hazard propagation using FRDL models is provided in Reference [18]. As described in this guidance, the FRDL formalism enables the analysis of the behavioural mechanisms through which hazard propagate through the system architecture, and as a result can be used to determine both the effects of hazards (as traditionally required by FHA tables) as well as potential causes (which can support the creation of other types of hazard tables).

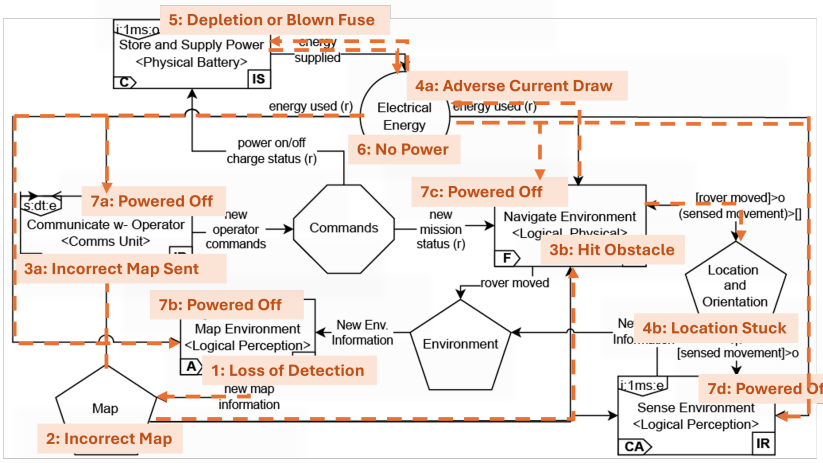


Figure 3: Propagation of *Complete Loss of Obstacle Detection* in TROUPE Functional Architecture

An example illustrating how FRDL models are used to trace the propagation of the failure condition *Complete Loss of Obstacle Detection* in the *Map Environment* function is shown in Figure 3. As depicted, this failure leads to the generation of incorrect maps, which in turn causes the *Communicate with Operator* function to transmit inaccurate information to the ground station—potentially compromising the mission. Additionally, the incorrect map may cause the *Navigate Environment* function to steer the rover into obstacles, resulting in a crash. One might choose to stop the analysis at this point, assuming that a crash would lead to the complete loss of the rover. However, if there is a possibility that the rover could continue functioning after a crash, further tracing of the failure propagation can reveal critical insights into the potential for post-crash recovery. For instance, assuming the *Navigate Environment* function lacks crash detection capabilities, the crash might cause the wheels (contained within the Navigate Environment function) to continue spinning without movement. This behaviour could result in unintended current draw, leading to a depleted battery or blown fuse in the *Supply Power* function. Ultimately, this would cause the power-dependent functions of the rover to lose power, leaving it trapped in its environment and unable to perceive its environment or even communicate with an external operator.

It should be noted that hazard models such as that shown in Figure 3 require validation and refinement to ensure that the model can be used to adequately analyze hazards. In this case, the model was created in a collaboration with the developers of the system, who were interested in modeling the autonomy pipeline, leading to a model that emphasized the perception, navigation, control, and communication functions over the physical hardware interactions. While the developers agreed that this hazard model was valid for the use-case of this study, extension to other use-cases would require further definition of the relevant functions, flows and behaviors. To use this process in a new system, the analysts would thus need to confer with technical experts to ensure that the relevant systems related to the hazards under study are adequately represented.

(ii) Hazard Elicitation from Historic Data

To assist and augment the expert-driven FRDL-based FHA process, hazards may additionally be elicited from historic data. The main benefit of this is to improve the coverage of the hazards elicited (i.e., to identify relevant hazards that may have otherwise been left out), but this step may also provide details that may help refine the analysis or provide relevant past examples or lessons learned from prior projects that will enhance the analysis. MIKA provides both high-level thematic hazards that could prompt further engineering analysis or inquiry as well as concrete,

specific examples of past lessons learned or incidents that may be relevant to the specific system being designed. Because the information provided is historical in nature, it requires engineering judgment and analysis before inclusion in the analysis. The information used to elicit hazards can be drawn from the same and similar systems, or even from other domains [23], and is reviewed for relevance to the system being designed. For instance, consider a scenario in which an analyst reviews lessons learned from autonomous driving when performing a hazard assessment for an autonomous rover. A hazard that may be identified from historic data on autonomous driving is driver complacency. An analogous hazard for an autonomous rover that may be considered for the analysis would be operator complacency. A similar hazard exists in both domains, but in this case the role of the operator/driver differs. In this way, historic lessons learned and incidents are used to elicit (rather than generate) hazards for the analysis.

The general process for eliciting hazards from historic data using MIKA is as follows, as described in detail in prior work [24,25]:

(i) Identify relevant datasets.
(ii) Extract high-level thematic hazards from datasets.
(iii) Search the dataset for specific details on thematic hazards or other topics of interest.

We elicited hazards for TROUPE using historical lessons learned described in NASA's Lessons Learned Information System (LLIS) [43]. Several high-level thematic hazards were identified from past space missions, including issues related to batteries, actuators, and communications systems. These high-level themes can be formulated as queries used to search the set of documents. For example, we searched "communication loss" to retrieve reports on specific radio or communications failures. Additional queries may be added based on the concept of operations, the functional description of the system, or otherwise as needed during the hazard analysis. Using this approach, multiple hazards were identified or refined. One example is a software issue in which a rover becomes unable to perform any task requiring computer memory. Another issue identified is harsh terrain causing mechanical damage to the over which could affect its ability to complete its mission. Each of these are lessons learned from past missions and, as part of the hazard analysis, may be considered or adapted to TROUPE's specific mission and architecture. This step can improve coverage of hazards that may have been missed in Steps 0-2, provide additional details on hazard causes or effects, and provide references to historical examples of similar or related failures or identified issues.

(iii) Hazard Simulation

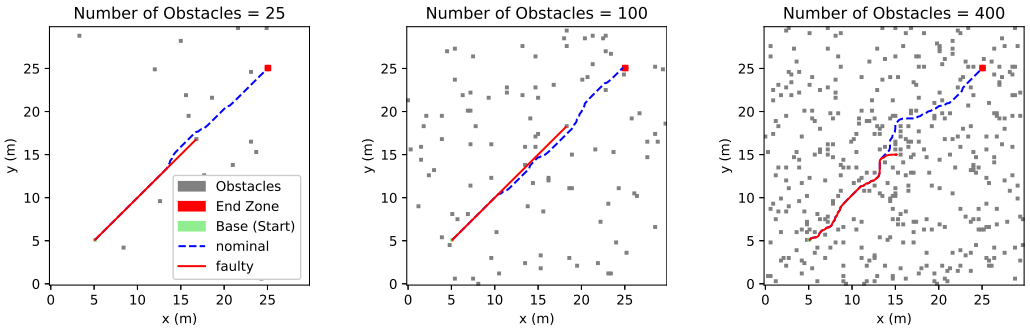
The simulation step in CFHA employs scenario-based simulations to refine and validate the hazard assessments from previous steps and to derive detailed safety requirements. These simulations are implemented using fmdtools, where behaviors for each function and flow in the FRDL model (as discussed in Section (b)) are specified and executed across multiple scenarios. Scenarios can include variations in parameters, single or multiple fault injections, or combinations of these. Variables that remain constant throughout the simulation are passed as parameters to the model. As shown in Table 1 three categories of parameters were defined for the TROUPE rover. Mission parameters affect the rover's performance during the mission. Ground control parameters are ground station command inputs provided to the rover at the start of the mission. Finally, environmental parameters define the grid on which the rover operates. The values or the distributions that were used for each of the parameter are listed in Table 1.

Behaviours for each failure condition associated with each function are defined in fmdtools to enable fault injection and propagation. For example, the six failure conditions listed in Table 3 were modeled in fmdtools as failures of the function *Map Environment*. One example of a fault scenario that can be injected into the simulation is *Complete loss of obstacle detection*. The simulation then runs with this fault until an end condition, such as the rover arriving at its destination, colliding with an obstacle, or depleting its power supply, is satisfied.

Table 1: Hazard Simulation Parameters

Category	Parameter	Description	Value
Ground Control	Destination	The coordinates of the destination	(25x25 m)
	Speed	Maximum Speed of the Rover	10 cm/s
Mission	Vision Range	The range of the perception sensor (e.g., camera, lidar)	5 m
	Time Scale	Size of a timestep	1 s
	Maximum Ghost Points	The maximum number of ghost points the mapping function will perceive per time-step when "Ghost Obstacle Detection" failure condition is present	1
Environment	Grid Size	Number of grid points in x and y direction creating a x*y grid	100
	Block Length	Length of each of the grid points	30 cm
	Number of Obstacles	Number of Obstacles in the Grid (used to set the obstacle density in the environment)	25/100/400

To study the *loss of obstacle detection* fault further, it was instantiated in the simulation in environments with a low, medium, and high density obstacles in randomly-generated obstacle distributions, as shown in Figure 4. As shown, in its worst case, a lack of obstacle detection can cause the rover to collide with an obstacle. This worst-case scenario is more likely to occur in environments with a high density of obstacles because obstacles are proportionally more likely to be in the rover's path of travel.



(a) Low Density Scenario (b) Medium Density Scenario (c) High Density Scenario

Figure 4: Worst-Case Obstacle Detection Faults Across Varying Environments.

To prevent the *lack of obstacle detection* fault from resulting in crash, a threshold could be placed on the time during which no new obstacles are detected. If this threshold is below the vision range in that time ($50s$ to reach the $5m$ vision range at the rovers max speed of $10cm/s$), the rover could then be put in a standby mode to prevent collision. However, depending on the density of objects in the environment, there is a likelihood that this mode will be entered simply due there not being obstacles in its path of travel. To thus evaluate whether this threshold is adequate, the rover was simulated over each low, medium, and high density scenario to generate a sample of 1000 time-intervals between obstacle detection. These distributions are shown in Figure 5. As shown, in medium and high density scenarios, the threshold of $50s$ is never reached (over 1000 intervals), meaning that it is expected to perceive an obstacle within $50s$ of perceiving the last obstacle. Thus, in these scenarios, a requirement of putting the rover in standby after not perceiving obstacles can reasonably be set without a significant expectation of false positives. However, in the low density scenario, there is a substantive non-zero probability that the rover will not perceive an object in $50s$ due to the low density of objects in the environment.

Table 2 shows the calculated probability of a collision in a route given a certain obstacle density, along with the simulated frequency of no obstacles being detected in an interval of over 50 seconds. As shown, in the medium and high density scenarios, the probability of a

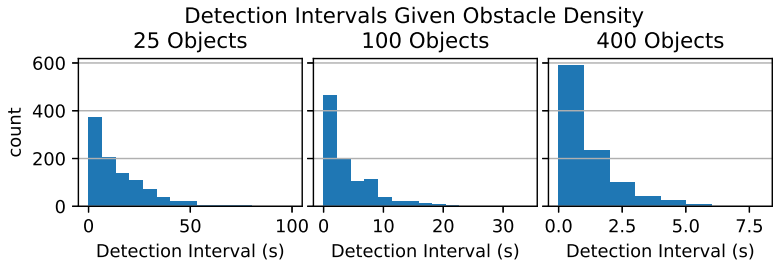


Figure 5: Study of obstacle detection times given environmental obstacle density

Obstacle Density:	**Low**	**Medium**	**High**
Calculated Probability of Collision Over Route	0.19	0.57	0.97
Simulated Frequency of No Obstacles detected in a $\geq 50s$ interval	0.037	≈ 0	≈ 0

Table 2: Summary Statistics Affecting Implementation of Object Detection Threshold. ≈ 0 indicates that no intervals of $\geq 50s$ without obstacle detection were observed across 1000 simulated samples, indicating that such intervals are highly unlikely but not theoretically impossible under the given obstacle density.

collision increases while there were no simulations in which the frequency of obstacles detected had an interval $< 50s$ (making these outcomes unlikely but not impossible). In the low density scenario, there is still a significant probability of collision over the route (0.19), however, there is also a substantial likelihood of no obstacles being detected in a $50s$ interval. As such, it would be advantageous to provide additional functionality to avoid these errors–such as system checks, fault diagnosis, re-booting, and/or activating redundant environmental sensors after being placed into standby–to determine whether there was actually a *lost obstacle detection* fault or if there were simply no objects in the immediately explored environment.

(d) Define Mitigations and Requirements

This step leverages the results of the hazard analysis to derive requirements that mitigate the identified hazards. These requirements and mitigations may encompass design selection, design modifications, engineered safety features, safety devices, warning systems, or operational procedures and training. Traditionally, this has been an expert-driven process focused primarily on identifying high-level safety objectives and abstract "functional" requirements that do not readily formalize. In contrast, the integration of scenario-based simulations within the CFHA process enables the elicitation of more granular and detailed requirements.

For the purposes of this paper, the mapping function of the TROUPE rover was selected as the basis for requirement elicitation. This effort resulted in eighteen total requirements, including eight high-level requirements. Table 4 presents a representative subset of requirements derived to mitigate the *Complete Loss of Obstacle Detection* failure condition of the *Map* function.

Requirement 1, "The failure probability for obstacle detection shall be no less than 10^{-3}," exemplifies a high-level safety requirement identified through traditional expert-driven analysis without simulation support. Such requirements establish overall system reliability targets but provide limited guidance for detailed design implementation.

In contrast, Requirements 2, 2.1, and 2.2 demonstrate the value of simulation-based requirement elicitation. Requirement 2 establishes the need for continuous monitoring: "Rover shall evaluate if new obstacles are detected at each timestep." Building upon this foundation, Requirements 2.1 and 2.2 specify concrete operational behaviors derived from resilience simulation studies. Requirement 2.1 addresses early warning by stipulating that "When no new obstacles are detected in the map for 8 seconds and maps continue to generate the rover shall send

Table 3: Functional Hazard Assessment for Function *Map* When the Rover is in the Mapping Phase

Failure Condition	**Causes of Failure Condition**	**Failure Effects**
Complete loss of obstacle detection	Invalid/no inputs due to loss of sensors, poor physical/software connection to the feed, software unable to initialize obstacles, software unable to recognize obstacles, loss of/incorrect localization	Path plans with obstacles can result in collision with obstacles resulting in loss of rover, incomplete mission due to not reaching destination
Partial loss of obstacle detection	Invalid inputs due to loss of sensors, poor physical/software connection to the feed, software unable to initialize obstacles, software unable to effectively filter out sensor noise, software unable to recognize obstacles, loss of/incorrect localization	Path plans with obstacles can result in collision with obstacles resulting in loss of rover, incomplete mission due to not reaching destination
Non-existent obstacle detection	Loss of/incorrect localization, invalid inputs due to physical/software connection interference, software initializes obstacles incorrectly, software fails to filter out sensor noise	Path plans that are suboptimal resulting in rover traversing unwanted areas delaying or possibly compromising the mission. The rover may consume additional energy, leaving little energy left for future missions.
Loss of mapping input data	Loss/invalid sensor data, poor physical/software connection, loss of power supply to sensor/onboard mapping electronics	Rover may crash into obstacles resulting in loss of rover. Mission will not reach destination and loose all mapping functions, resulting in failed mission.
Loss of mapping history	Failed/malfunctioning map storage component caused by physical damage or electrical damage.	Loss of map retention resulting rover not being able to use maps of already mapped areas. This may require the rover to re-map these areas, resulting additional energy consumption leading to no having sufficient energy to return to base, resulting in a failed mission and additional effort to retrieve the rover.
Delay in building map	Low energy supply, poor computational resource allocation, computational resource limitation, latency in the sensor feed, poor parameter tuning	Delayed mission due to rover waiting for new maps. Incomplete mission due to additional energy used waiting for updated maps. Potential collision with obstacles and loss of rover due to delayed detection of obstacles.

a warning to ground station," including specific diagnostic information (time since last detection and distance to closest object). Requirement 2.2 defines a fail-safe action: "Rover shall disengage from auto navigation if no new obstacles are detected in the map for 50 seconds of time."

The specific time thresholds in Requirements 2.1 and 2.2 are not fixed values but must be calibrated to the expected operating environment. As shown in Section iii, the distribution of inter-detection intervals varies significantly with obstacle density: sparser environments produce longer nominal intervals, increasing the risk of false positives if thresholds are set too tightly. Historically, such operationally grounded requirements would emerge only late in the design process, once sufficient testing data are available. Deriving them early (through simulation rather than empirical testing) enables mitigative measures to be incorporated into the system architecture from the outset, rather than retrofitted into a mature design, thereby reducing both cost and the risk of undetected hazardous conditions.

Concretely, given the rover's maximum speed of 10 cm/s (Table 1), a rover traveling at full speed would cover its full 5 m sensor range in 50 seconds. This value therefore serves as an absolute upper bound for Requirement 2.2, i.e., if no new obstacle has been perceived within 50 seconds, the rover cannot guarantee it would detect an obstacle before collision, and autonomous navigation should be disengaged. For the early-warning threshold of Requirement 2.1, the simulation results in Figure 5 indicate that in dense environments no nominal inter-detection interval approached 50 seconds across 1000 sampled intervals, suggesting that a tighter threshold of 8 seconds can reliably signal a probable fault in such conditions while avoiding an unacceptable false-positive rate.

Table 4: Derived Requirements for mitigating the Loss of Obstacle Detection Failure Condition

ID	Requirement Derived From CFHA	Revised Requirements Based on FRETish feedback	Formal Requirements Specified Using FRETish
1	The failure probability for obstacle detection shall be no less than 10^{-3}.		`The rover` shall `with a probability <` 10^{-3} `never` satisfy `failedObstacleDetection`
2	Rover shall evaluate if new obstacles are detected at each timestep.		Decomposed into 2.1 and 2.2.
2.1	When no new obstacles are detected for 8 seconds and maps continue to generate the rover shall send a warning to ground station. The warning shall include the amount of time since last obstacle was detected and the distance to last detected closest object.	When no new obstacles are detected in the map for 8 seconds and maps continue to generate the rover shall send a warning to ground station. The warning shall include the amount of time since last obstacle was detected and the distance to last detected closest object.	`whenever generateMaps upon persisted(7, noNewObstaclesInMaps) the rover` shall `immediately` satisfy `sendWarningToGS`
2.2	Rover shall stop operations if no obstacles are detected for 50 amount of seconds	Rover shall disengage from auto navigation if no new obstacles are detected in the map for 50 seconds of time.	`upon persisted(49,noNewObstaclesInMaps) rover` shall `immediately` satisfy `disengageFromAutoNavigation`

(e) Formalize Requirements

Following the enhanced hazard analysis (Step 4) and the definition of mitigations and requirements (Step 5), the identified requirements must be formalized to enable rigorous analysis and eventual runtime monitoring. This formalization step is performed using FRET, which transforms requirements written in structured natural language into mathematical formalisms suitable for verification and validation activities.

The formalization process begins with expressing each requirement in FRETISH, FRET's restricted natural language. FRETISH enforces a specific grammatical structure that eliminates common ambiguities found in traditional natural language requirements while remaining intuitive for domain experts. Each FRETISH requirement follows the pattern: *scope* - *condition* - *component* - *shall* - *probability* - *timing* - satisfy *response*, where each element has precise semantic meaning [31,44]. Only the *component*, *shall*, and satisfy *response* fields are mandatory. The remaining fields are optional and are instantiated only when the requirement demands them.

The *probability* field was introduced in the probabilistic extension of FRETISH [44] to support autonomous systems, where uncertainty in perception, actuation, and environment means that deterministic guarantees are often unachievable. This field allows engineers to express bounds on the likelihood that a condition holds over system executions. Requirement 1 uses this extension: rather than asserting that obstacle detection will never fail, it specifies a bound on the probability of failure.

Table 4 illustrates the requirement formalization process through three columns. The second column shows the original requirements as elicited from the hazard analysis. The third column presents revised versions that incorporate feedback from FRET's interactive editing interface, which refines the natural language to align with FRETISH grammar while preserving semantic intent. The fourth column shows the final FRETISH specification. Not all rows have entries in every column: Requirement 1 has no revised version because its natural language formulation required no clarification before formalization. Furthermore, Requirement 2 has no FRETISH

specification because it is a high-level requirement decomposed into Requirements 2.1 and 2.2, which are themselves formalized individually.

Two changes in the revised column are worth noting. First, Requirement 2.1 was refined from "no new obstacles are detected" to "no new obstacles are detected in the map": a subtle but important scoping change that ties the condition to the mapping function specifically rather than to raw sensor input. Second, Requirement 2.2 was revised from the vague "stop operations" to "disengage from auto navigation", which identifies the precise system response required and avoids ambiguity about which operations are affected.

The fourth column displays the final FRETISH specification for each requirement. These specifications leverage FRETISH's precise semantics [33]. For instance, Requirement 2.1 is specified as: "`whenever generateMaps upon persisted(7, noNewObstaclesInMaps) the rover` shall `immediately` satisfy `sendWarningToGS`" This specification captures several critical semantic elements: the *whenever* condition indicates the requirement applies throughout map generation operations; the `upon persisted(7, noNewObstaclesInMaps)` condition specifies that the hazardous state must persist for 8 seconds; and the `immediately` timing requires the warning to be sent without delay once the condition is met. Similarly, Requirement 2.2 uses the `upon persisted(49, noNewObstaclesInMaps)` pattern to specify the duration threshold before automatic disengagement. Notice that Requirement 2 is not formalized in FRETISH since it is a high-level requirement that is decomposed into two (2.1 and 2.2). Requirement 1 is formalized in FRETISH using the probabilistic FRETISH extension, which was added specifically for the specification of requirements for autonomous systems [44].

FRET provides immediate feedback during requirement entry through its interactive editor, which displays grammar hints, example requirements, and real-time semantic interpretations. For each FRETISH requirement entered, FRET automatically generates multiple formal representations, including formulae in Linear Temporal Logic (LTL) [45] and Probabilistic Computation Tree Logic (PCTL) [45]. Probabilistic logics become increasingly important within the context of autonomous systems since probabilities are often used to capture uncertainty. Additionally, FRET supports interactive simulation of the formalized requirements, allowing engineers to validate that the formal semantics accurately capture the intended behavior before proceeding to more comprehensive analysis activities.

The FRDL model developed in Step 1 provides essential context for requirement formulation, as it clearly delineates the functions, flows, and interactions that requirements must constrain. This model-based traceability ensures that each requirement can be mapped back to specific hazards (e.g., *Complete Loss of Obstacle Detection* in Table 3), system functions (e.g., the *Map* function), and ultimately to verification activities.

(f) Analyze Requirements

Once requirements have been formalized in FRET (Step 6), they undergo comprehensive analysis to identify potential conflicts and inconsistencies before proceeding to detailed design and implementation phases.

FRET supports realizability analysis [38] to determine whether a given set of requirements can be simultaneously satisfied by any implementation. Unrealizable requirements indicate fundamental conflicts in the specification that must be resolved during design. For example, if Requirements 2.1 and 2.2 specified conflicting time thresholds or incompatible system responses, realizability analysis would detect this inconsistency and flag it for resolution.

The analysis results from FRET result in requirement repair based on identified issues. Conflicts detected during analysis may necessitate revisiting the hazard analysis (Steps 2-4) or adjusting proposed mitigations (Step 5). This iterative refinement process continues until the requirement set is confirmed to be consistent, realizable, and complete with respect to the identified hazards.

In our case study, realizability analysis revealed an issue between Requirements 2.1 and 2.2. Specifically, since the condition triggering disengagement (50 seconds without

obstacle detection) is a strict superset of the condition triggering the warning (8 seconds without obstacle detection), disengagement can only occur after the warning threshold has already been crossed. Yet, no requirement explicitly ensured that the warning state was maintained or acknowledged before disengagement. To resolve this issue, an additional bridging requirement was introduced: "`upon persisted(49, noNewObstaclesInMaps) the rover` shall `before disengageFromAutoNavigation` satisfy `sendWarningToGS` ." This example illustrates the value of formal realizability checking as an integral step in the workflow: conflicts that might otherwise surface only during late-stage testing or, worse, during operation, are instead caught and resolved early in the design process.

The formalized requirements that emerge from this process provide a solid foundation for subsequent verification activities, including the generation of runtime monitors that can detect requirement violations during system operation. For the TROUPE rover, runtime monitors based on Requirements 2.1 and 2.2 would continuously evaluate whether new obstacles are being detected in generated maps and trigger appropriate warnings or safety actions when the specified time thresholds are exceeded. Through its connection to analysis tools [36,46] FRET enables the automatic generation of verification code and runtime monitors directly from the formalized requirements, ensuring end-to-end traceability from hazard identification.

While the discussion above focuses on Requirements 1, 2.1, and 2.2 to illustrate the formalization process in detail, these are representative of a broader analysis. In total, eighteen requirements were derived for the mapping function of the TROUPE rover, spanning six failure conditions: complete loss of obstacle detection, partial loss of obstacle detection, non-existent obstacle detection, loss of mapping input data, loss of mapping history, and delay in building a map (Table 3). Most of the requirements are operational requirements of the form of Requirements 2.1 and 2.2, specifying concrete system behaviors triggered by detectable conditions. Across this full set, the formalization process in FRET prompted natural language clarifications in the majority of requirements. These were analogous to the scoping and action refinements illustrated in Table 4. The realizability analysis identified one ordering underspecification, resolved by adding requirement, with no further conflicts detected in the final requirement set.

4. Discussion

(a) What the integration between FRET and HADES provides

While HADES and FRET are both existing NASA tools, this is the first time that they are combined in a hazard assessment workflow. Note that the information produced by one tool shapes the inputs and outputs of the other in ways that neither tool could achieve alone:

HADES informs what FRET must specify. Without simulation, a requirements engineer faces the problem of threshold selection with no principled basis for specific values. For example, the "50 seconds" threshold in Requirement 2.2 cannot be derived from expert judgment alone: without the fmdtools resilience simulation (Section 3.iii), the engineer must either leave the requirement underspecified (e.g., "the rover shall stop if no obstacles are detected for a long time") or defer the value to late-stage hardware testing. Both options are costly: an underspecified requirement cannot be formally analyzed or verified, while a deferred threshold means that conflicts and inconsistencies in the requirement set might go undetected until testing or operation, when they are far more expensive to resolve.

FRDL modeling shapes what MIKA queries are meaningful and MIKA expands the space of hazards that FRET must cover. The functional architecture developed in Step 1 of Figure 1 defines the semantic categories (e.g., functions, flows, failure conditions) against which MIKA queries are formulated. A query for "communication loss" is meaningful only once the Communicate With Operator function has been identified as a first-class element of the system architecture. The FRDL model provides the ontological frame that makes MIKA's results actionable. Conversely, MIKA surfaces hazard classes that the expert-driven FRDL analysis alone

might have missed. In the TROUPE case study, memory-related software faults and terrain-induced mechanical damage were identified from NASA LLIS and not from the initial functional model. Each such hazard, once incorporated into the hazard table, becomes a new source of requirements that FRET must eventually specify and verify. MIKA thus directly expands the scope of the formal specification.

FRET analysis feeds back into hazard analysis. The realizability conflict identified in Section 3.(f) was not apparent from reading the FRDL model or the simulation results. It emerged only from FRET's realizability analysis. Resolving it required introducing a bridging requirement that, in turn, constitutes a new constraint on the system's behavior that was not present in the original hazard table. Thus, formal requirements analysis can reveal gaps in the hazard analysis itself.

(b) Lessons Learned

The application of the proposed hazard assessment workflow to the TROUPE rover case study yielded several important lessons that can inform future applications of this methodology to other increasingly autonomous systems.

NLP-based hazard elicitation complements but does not replace expert judgment. Querying NASA's LLIS with MIKA both confirmed hazards already identified by experts and surfaced additional failure modes, such as memory-related software faults and terrain-induced mechanical damage, as concrete examples of lessons learned from past missions that are relevant to TROUPE's mission and architecture. However, historical data cannot be incorporated directly: they still need to be reviewed by a domain expert before being included in the hazard analysis. This review step is not a limitation of the approach but an inherent property of reasoning from analogous systems and past missions. The value of automated elicitation lies precisely in broadening the set of candidates for expert consideration, which reduces the risk that a relevant hazard class is completely missed.

Simulation enables the early derivation of threshold-based requirements. The scenario-based resilience simulations conducted using fmdtools produced quantitative evidence about hazard likelihood that typically would have been unavailable through expert elicitation alone. For instance, simulating the *Complete Loss of Obstacle Detection* fault across low, medium, and high obstacle density environments revealed that a 50-second detection threshold reliably signals a fault in medium and high density environments, but that in sparse environments the rover may plausibly go 50 seconds without detecting an obstacle even under nominal conditions. This density-dependent behaviour directly shaped Requirements 2.1 and 2.2, and informed the recommendation that additional diagnostic functionality, such as fault diagnosis or redundant sensor activation, be considered for low-density operating scenarios. Such threshold-based requirements would typically emerge only during late-stage testing, when more design details are available and detailed empirical studies are feasible. Identifying them early in the design process enables mitigations to be incorporated into the system architecture at an early stage.

Formal requirements analysis catches specification issues early. The realizability analysis performed in FRET identified a latent conflict between Requirements 2.1 and 2.2 that was not apparent from reading the natural language versions: disengagement could be triggered without the preceding warning being explicitly guaranteed. Resolving this required introducing a bridging requirement, a refinement that would realistically have been deferred to late-stage integration testing or discovered only during operation without the realizability checking step. More broadly, the FRETISH formalization process itself proved valuable independently of the realizability analysis. Expressing requirements in a structured grammar with precise semantics forced clarifications in natural language that improved the requirements before any formal tool was invoked (e.g., "stop operations" becoming "disengage from auto-navigation," and "no new obstacles detected" being scoped to "in the map"). Introducing precise requirement specification and analysis early in the design process can help identify and resolve specification conflicts before they propagate into implementation, reducing the cost and risk of late-stage re-designs.

(c) Limitations

While the proposed methodology shows promise, several limitations must be acknowledged.

Single-rover scope. The current case study focuses on a single TROUPE rover rather than the complete system-of-systems including multiple rovers, the ground station, and the charging base. While a comprehensive assessment would eventually need to address inter-rover interactions and emergent swarm behaviors, focusing on a single rover enables a more tractable demonstration of the methodology. Many of the most safety-critical behaviors of the system (e.g., obstacle avoidance, localization, mapping, and power management) are properties of individual rovers, and the hazards and requirements derived here would remain relevant even in a multi-rover analysis. Furthermore, the methodology itself is not limited to single-agent systems because HADES and FRET explicitly support system-of-systems modeling and specification.

Modeling assumptions. The simulation model assumes uniformly distributed obstacles, which is unlikely to hold in a real planetary or terrestrial environment. However, the primary contribution of the simulation is not the specific threshold values derived (which would need to be recalibrated for a real deployment environment) but rather the demonstration that simulation can reveal the *existence* of density-dependent threshold behavior and drive the derivation of more specific requirements than expert elicitation alone would produce. More realistic environment models would sharpen the quantitative outputs without changing the methodology itself.

Toolchain integration. In the current demonstration, data transfer between HADES and FRET is performed manually. While this might introduce friction in practice, it did not prevent the methodology from being applied, and the manual process made the data flows between tools explicit and auditable, which has its own value during early-stage methodology development. Toolchain integration is an engineering effort that can proceed independently of the methodology itself. The interfaces between tools are well-defined, and the manual process demonstrated here provides a clear example of what an automated integration would need to support. Finally, the current demonstration does not include a controlled comparison against a conventional FHA-only process. Evaluating the workflow's relative benefits in a systematic way remains an important direction for future work.

5. Related Work

Resilience simulation in fmdtools was originally designed to enable the parametric exploration and optimization of resilience to perform trade studies in conceptual design [28,47]. While conceptual design heavily informs high-level requirements, no systematic process had been formulated to explicitly formalize those requirements from simulation outputs. NLP-based hazard elicitation and resilience simulation have previously been combined in a wildfire response case study [48], and a subsequent demonstration applied fmdtools, FRDL, and MIKA jointly [49]. However, neither work outlined a systematic methodology for integrating these tools into a larger hazard analysis process, nor showed how their combined outputs could inform formal requirements specification. The present work addresses both gaps by introducing FRDL as a structural bridge between elicitation and simulation, and by connecting the resulting hazard analysis to FRET for requirements formalization and analysis.

NLP and large language models (LLMs) have been applied to hazard elicitation [50,51] and to informing subsequent system modeling [25,52] but these contributions have typically been conducted as standalone activities, without extension to simulation or resilience studies. Formal requirements specification for autonomous systems has similarly been studied in isolation across several tools [53]. Most approaches formalize requirements that have already been elicited, without addressing the elicitation problem itself. The novelty of our work lies in the integration of all three activities, i.e., hazard elicitation, resilience simulation, and formal specification, within a single workflow, where simulation results from fmdtools supply the structured, quantitative inputs that make threshold-based requirements formalizable rather than left as placeholders.

6. Conclusion

This article presented a novel hazard assessment workflow for the assurance of increasingly autonomous systems, demonstrating how the integration of computational tools can enhance traditional safety analysis approaches. The proposed methodology leverages two NASA-developed toolsets, i.e., HADES and FRET, to address the challenges posed by autonomous systems operating in complex, dynamic environments with minimal human oversight.

The application of this methodology to the TROUPE rover case study demonstrated its practical utility. Through the systematic eight-step process, we developed a functional model of the rover, conducted preliminary hazard analysis, augmented this analysis with historical data from NASA's Lessons Learned Information System, and performed simulation studies to identify critical operational thresholds. The resulting requirements, formalized in FRETISH language, provide unambiguous specifications that can be directly transformed into temporal logic formulae and runtime monitors. The traceability maintained throughout this process, from initial hazard identification through requirement formalization and analysis, ensures that safety considerations are systematically addressed and documented.

References

1. Trevelyan J, Hamel WR, Kang SC. 2016 pp. 1521–1548. In *Robotics in Hazardous Applications*, pp. 1521–1548. Cham: Springer International Publishing. (10.1007/978-3-319-32552-1$_5$8)
2. Bruzzone AG, Longo F, Agresta M, Di Matteo R, Maglione GL et al.. 2016 Autonomous systems for operations in critical environments. *Simulation Series* pp. 17–24.
3. Mbaye S, Jones G, Davies MD. 2023 Analysis of input from wildfire incident experts to identify key risks and hazards in wildfire emergency response. In *AIAA SCITECH 2023 Forum* p. 2710. (10.2514/6.2023-2710)
4. Sinha S, Lee S, Singh S. 2026 Survey on Reconnaissance Autonomous Robotic Systems for Disaster Management. *Sensors (Basel, Switzerland)* **26**, 1659.
5. Truszkowski W, Hinchey M, Rash J, Rouff C. 2006 Autonomous and autonomic systems: a paradigm for future space exploration missions. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* **36**, 279–291. (10.1109/TSMCC.2006.871600)
6. Tuncali CE, Fainekos G, Prokhorov D, Ito H, Kapinski J. 2020 Requirements-Driven Test Generation for Autonomous Vehicles With Machine Learning Components. *IEEE Transactions on Intelligent Vehicles* **5**, 265–280. (10.1109/TIV.2019.2955903)
7. Pinto A. 2021 Requirement Specification, Analysis and Verification for Autonomous Systems. In *2021 58th ACM/IEEE Design Automation Conference (DAC)* pp. 1315–1318. (10.1109/DAC18074.2021.9586208)
8. SAE S-18 Committee. 1996 ARP4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment. SAE International.
9. Sun R, Zhong D, Li W, Lu M, Ding Y, Xu Z, Gong H, Zha Y. 2020 A Safety Analysis Method of Airborne Software Based on ARP4761. *Journal of Physics: Conference Series* **1673**, 012045. (10.1088/1742-6596/1673/1/012045)
10. Committee SC-205. 2011 Software Considerations in Airborne Systems and Equipment Certification. Technical Report DO-178C RTCA, Inc.
11. S-18 Aircraft And System Development And Safety Assessment Committee. 2023 Guidelines for Development of Civil Aircraft and Systems. Aerospace Recommended Practice ARP4754 Rev. B.
12. Dezfuli H, Benjamin A, Everett C, Feather M, Rutledge P, Sen D, Youngblood R. 2015 NASA system safety handbook. Volume 2: System safety concepts, guidelines, and implementation examples. Technical Report SP-2014-612 National Aeronautics and Space Administration.
13. Koopman P, Wagner M. 2016 Challenges in autonomous vehicle testing and validation. *SAE International journal of transportation safety* **4**, 15–24. (10.4271/2016-01-0128)
14. Alexander R, Herbert N, Kelly T. 2009 Deriving safety requirements for autonomous systems. In *4th SEAS DTC technical conference* pp. 1–8.
15. Gervasi V, Ferrari A, Zowghi D, Spoletini P. 2019 pp. 191–210. In *Ambiguity in Requirements Engineering: Towards a Unifying Framework*, pp. 191–210. Cham: Springer International Publishing. (10.1007/978-3-030-30985-5$_1$2)

16. Ribeiro Q, Ribeiro M, Castro J. 2022 Requirements engineering for autonomous vehicles: a systematic literature review. pp. 1299–1308. (10.1145/3477314.3507004)
17. Benz NA, Sljivo I, Woodard A, Vlastos PG, Carter CK, Hejase M. 2024 The Troupe System: An Autonomous Multi-Agent Rover Swarm. In *AIAA SCITECH 2024 Forum* p. 2894. (10.2514/6.2024-2894)
18. Mbaye S, Hulse DE, Irshad L, Walsh HS, Andrade SR. 2025 Towards Computational Functional Hazard Assessment (CFHA): A Gap Analysis and Concept for Emerging Aviation Systems. In *AIAA SCITECH 2025 Forum* p. 1411. (10.2514/6.2025-1411)
19. Mbaye S, Hulse DE, Irshad L, Walsh HS, Andrade SR. In Press Computational Functional Hazard Assessment (CFHA): A New Paradigm for Autonomous Aviation Systems. *Journal of Aerospace Information Systems*.
20. Andrade SR, Walsh HS. 2023 Discovering a failure taxonomy for early design of complex engineered systems using natural language processing. *Journal of Computing and Information Science in Engineering* **23**, 031001. (10.1115/1.4054688)
21. Walsh HS, Andrade SR. 2022 Semantic Search With Sentence-BERT for Design Information Retrieval. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* vol. 86212 p. V002T02A066. American Society of Mechanical Engineers. (10.1115/DETC2022-89557)
22. Andrade S, Walsh H. 2023 MIKA: Manager for Intelligent Knowledge Access Toolkit for Engineering Knowledge Discovery and Information Retrieval. In *INCOSE International Symposium* vol. 33 pp. 1659–1673. Wiley Online Library.
23. Irshad L, Walsh HS. 2025 Learning from accident reports using language models: human errors and error mechanisms in aviation and railway accidents. *Journal of Computing and Information Science in Engineering* pp. 1–12. (10.1115/1.4070042)
24. Mbaye S, Walsh HS, Jones G, Davies M. 2023 BERT-based Topic Modeling and Information Retrieval to Support Fishbone Diagramming for Safe Integration of Unmanned Aircraft Systems in Wildfire Response. In *2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC)* pp. 1–7. IEEE. (10.1109/DASC58513.2023.10311309)
25. Mbaye S, Walsh HS, Davies M, Infeld SI, Jones G. 2024 From BERTopic to SysML: Informing Model-Based Failure Analysis with Natural Language Processing for Complex Aerospace Systems. In *AIAA SCITECH 2024 Forum* p. 2700. (10.2514/6.2024-2700)
26. Hulse D, Walsh H, Dong A, Hoyle C, Tumer I, Kulkarni C, Goebel K. 2021 fmdtools: A fault propagation toolkit for resilience assessment in early design. *International Journal of Prognostics and Health Management* **12**. (10.36001/ijphm.2021.v12i3.2954)
27. Irshad L, Hulse D. 2025 Toward Early Design Modeling and Simulation of Distributed Situation Awareness. *Journal of Computing and Information Science in Engineering* **25**, 061001. (10.1115/1.4067705)
28. Hulse D, Biswas A, Hoyle C, Tumer IY, Kulkarni C, Goebel K. 2021 Exploring architectures for integrated resilience optimization. *Journal of Aerospace Information Systems* **18**, 665–678. (10.2514/1.I010942)
29. Hulse D, Irshad L. 2023 Synthetic fault mode generation for resilience analysis and failure mechanism discovery. *Journal of mechanical design* **145**, 031707. (10.1115/1.4056320)
30. Hulse D, Mbaye S, Irshad L. 2025 The Functional Reasoning Design Language: Supporting Hazard Analysis With Graphical Models of Behavioral Interaction. *Journal of Mechanical Design* **147**, 101704. (10.1115/1.4069127)
31. Giannakopoulou D, Mavridou A, Pressburger T, Rhein J, Schumann J, Shi N. 2020 Formal Requirements Elicitation with FRET. In *International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ-2020)* number ARC-E-DAA-TN77785.
32. National Aeronautics and Space Administration. 2023 FRET: Formal Requirements Elicitation Tool. https://github.com/NASA-SW-VnV/fret.
33. Giannakopoulou D, Pressburger T, Mavridou A, Schumann J. 2021 Automated formalization of structured natural language requirements. *Information and Software Technology* **137**, 106590. (10.1016/j.infsof.2021.106590)
34. Jahier E, Raymond P, Halbwachs N. 2016 The lustre v6 reference manual. Technical Report v6.112.1 Verimag.
35. Mavridou A, Bourbouh H, Garoche PL, Giannakopoulou D, Pessburger T, Schumann J. 2020 Bridging the gap between requirements and simulink model analysis. In *International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ-2020)*.

36. Rozier KY, Schumann J. 2017 R2U2: Tool Overview. In Reger G, Havelund K, editors, *RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools, September 15, 2017, Seattle, WA, USA* vol. 3*Kalpa Publications in Computing* pp. 138–156. (10.29007/5pch)
37. Mavridou A, Katis A, Giannakopoulou D, Kooi D, Pressburger T, Whalen MW. 2021 From Partial to Global Assume-Guarantee Contracts: Compositional Realizability Analysis in FRET. In Huisman M, Păsăreanu C, Zhan N, editors, *Formal Methods* pp. 503–523 Cham. Springer International Publishing. (10.1007/978-3-030-90870-6$_2$7)
38. Katis A, Mavridou A, Giannakopoulou D, Pressburger T, Schumann J. 2022 Capture, Analyze, Diagnose: Realizability Checking Of Requirements in FRET. In *Computer Aided Verification: 34th International Conference, CAV 2022, Haifa, Israel, August 7–10, 2022, Proceedings, Part II* pp. 490–504. Springer. (10.1007/978-3-031-13188-2$_2$4)
39. Katis A, Mavridou A, Pressburger T. 2025 A Streamlined, Formal Approach to Requirements-Based Testing. In Dutle A, Humphrey L, Titolo L, editors, *NASA Formal Methods* pp. 159–179 Cham. Springer Nature Switzerland. (10.1007/978-3-031-93706-4$_1$0)
40. Sljivo I, Perez I, Mavridou A, Schumann J, Vlastos PG, Carter C. 2024 Dynamic Assurance of Autonomous Systems through Ground Control Software. In *AIAA SCITECH 2024 Forum* p. 1208. (10.2514/6.2024-1208)
41. McComas D. 2012 NASA/GSFC's Flight Software Core Flight System. In *Flight Software Workshop* vol. 11.
42. Kritzinger D. 2017 Chapter 3 - Functional Hazard Analysis. In Kritzinger D, editor, *Aircraft System Safety* , pp. 37–57. Woodhead Publishing. (10.1016/B978-0-08-100889-8.00003-9)
43. National Aeronautics and Space Administration NASA Public Lessons Learned System. https://llis.nasa.gov/. Accessed: 2025-12-09.
44. Mavridou A, Farrell M, Vázquez G, Pressburger T, Wang TE, Calinescu R, Fisher M. 2025 Automated Formalization of Probabilistic Requirements from Structured Natural Language. .
45. Baier C, Katoen JP. 2008 *Principles of model checking*. MIT press.
46. National Aeronautics and Space Administration. 2023 CoCoSim: Contract-based Compositional verification of Simulink models. https://github.com/NASA-SW-VnV/cocosim.
47. Hulse D, Hoyle C, Goebel K, Tumer IY. 2019 Quantifying the resilience-informed scenario cost sum: A value-driven design approach for functional hazard assessment. *Journal of Mechanical Design* **141**, 021403.
48. Andrade S, Hulse D, Irshad L, Walsh HS. 2022 Supporting hazard analysis for wildfire response using fmdtools and MIKA. Technical Report 20220014099.
49. Mbaye S, Hulse D, Irshad L, Walsh HS, Andrade SR. 2026 Computational Functional Hazard Assessment: A New Paradigm for Autonomous Aviation Systems. *Journal of Aerospace Information Systems* pp. 1–14.
50. Diemert S, Weber JH. 2023 Can Large Language Models Assist in Hazard Analysis?. In *Computer Safety, Reliability, and Security. SAFECOMP 2023 Workshops* pp. 410–422 Cham. Springer Nature Switzerland.
51. Ricketts J, Guo W, Pelham J, Barry D. 2025 Integrating an incident dataset with a question and answering language model to assist hazard identification: Comparison of an extractive and generative model. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability* **239**, 736–753.
52. Charalampidou S, Zeleskidis A, Dokas IM. 2024 Hazard analysis in the era of AI: Assessing the usefulness of ChatGPT4 in STPA hazard analysis. *Safety Science* **178**, 106608.
53. Lorch R, Meng B, Siu K, Moitra A, Durling M, Paul S, Varanasi SC, McMillan C. 2024 Formal methods in requirements engineering: Survey and future directions. In *Proceedings of the 2024 IEEE/ACM 12th International Conference on Formal Methods in Software Engineering (FormaliSE)* pp. 88–99.