Interlocking Safety Cases for Unmanned Autonomous Systems in Shared Airspaces

Michael Vierhauser, Sean Bayley, Jane Wyngaard, Wandi Xiong, Jinghui Cheng, Joshua Huseman, Robyn Lutz, and Jane Cleland-Huang

Abstract—The growing adoption of unmanned aerial vehicles (UAV) for tasks such as eCommerce, aerial surveillance, and environmental monitoring introduces the need for new safety mechanisms in an increasingly cluttered airspace. In our work we thus emphasize safety issues that emerge at the intersection of infrastructures responsible for controlling the airspace, and the diverse UAVs operating in their space. We build on safety assurance cases (SAC) – a state-of-the-art solution for reasoning about safety – and propose a novel approach based on interlocking SACs. The infrastructure safety case (ISAC) specifies assumptions upon UAV behavior, while each UAV demonstrates compliance to the ISAC by presenting its own (pluggable) safety case (pSAC) which connects to the ISAC through a set of interlock points. To collect information on each UAV we enforce a “trust but monitor” policy, supported by runtime monitoring and an underlying reputation model. We evaluate our approach in three ways: first by developing ISACs for two UAV infrastructures, second by running simulations to evaluate end-to-end effectiveness, and finally via an outdoor field-study with physical UAVs. The results show that interlocking SACs can be effective for identifying, specifying, and monitoring safety-related constraints upon UAVs flying in a controlled airspace.

Index Terms—UAV, Unmanned Autonomous Systems, Safety Assurance Cases, Monitoring

1 INTRODUCTION

An unmanned aerial system (UAS) or unmanned aerial vehicle (UAV), commonly known as a drone, is an aircraft without a human pilot onboard\(^1\) [1]. The increasing adoption of these systems for delivering goods, performing surveillance, conducting search and rescue operations, and supporting hobbyist activities [2], [3], [4], [5] introduces the need for safety mechanisms to be established at both the infrastructure and the UAV application level. In this context, the infrastructure reflects a specific operational environment, such as an urban area or a sporting event at which multiple UAVs, of different types, operate. Within this space, individual UAV operations leverage dedicated software systems, referred to as Ground Control Stations (GCS) to plan routes and manage UAV flights in ways that are consistent with governmental aviation rules [1], [6]. However, current GCS software provides little to no support for safely coordinating shared use of the public airspace. In recreational use, commonly accepted codes of conduct [6] may suffice to ensure safe operation.

However, as areas of commercial and community operations begin to emerge - where the use of multiple, potentially autonomous, UAVs are anticipated - additional means of controlling and managing UAVs within a shared airspace must be established to complement existing guidelines and regulations. In fact, the USA National Aeronautics and Space Administration (NASA) is currently developing the Unmanned Aerial Systems Traffic Management System (UTM) which is designed to manage low altitude airspace. UTM will provide capabilities for designing an airspace to include corridors and dynamic geofencing, to manage congestion, and to maintain UAV separation distances. It will also provide terrain information, and support for adaptive route planning [7].

A UAV flying in a shared airspace meets the definition of a safety-critical system whose “failure could result in loss of life, significant property damage, or damage to the environment”.

The number of incidents in which UAVs pose a threat to other aircraft or people continues to rise as their usage becomes more prevalent [8], [9]. The problem is exacerbated because an increasing number of commercial operators are using open source systems such as ArduPilot [10] to develop their own ground control systems without any experience of working in high-dependability domains.

The problem of UAV use in low-altitude shared airspaces is a multi-faceted one, in which acceptable levels of safety can only be achieved at the systems level by holistically considering the hardware, software, policy, and operator aspects of the shared airspace infrastructure and its interactions with potentially untrusted UAVs. In this paper, we explore safety hazards and mitigations for operating UAVs in a shared airspace with a particular emphasis on the role software, and consequently software engineering techniques, play in the process. Our solution explores the interplay of the airspace infrastructure and UAVs flying in the space. We build upon the use of safety assurance cases (SAC) which are increasingly being adopted as a means to describe and reason about system and software safety in

---

1. For the sake of simplicity, we refer to these types of systems as UAVs throughout the rest of the paper.

---

---
SACs provide formally structured arguments composed of claims, strategies, and evidence that a system is sufficiently safe for use. Evidence is diverse and may include formal models and proofs, simulation results, test case results, and informal artifacts such as best practices or training processes.

The challenge we seek to address is to provide support for software developers as they construct UAV applications even though they may lack formal training in safety practices, and to assure that their deployed UAV applications meet satisfactory safety standards. Our approach is meant to complement existing safety practices and could, for example, be used as part of a formal review process in a regulated domain. We propose a novel solution that splits a SAC into two different parts that can be effectively interlocked: an Infrastructure-level Safety Assurance Case (ISAC) and UAV-level pluggable Safety Assurance Cases (pSAC) for UAVs associated with it. We connect the individual UAVs’ safety assurance cases with the infrastructure safety case through safety interlocks. Furthermore we adopt the policy of trust but monitor, in which assumptions on behavior of UAVs specified in the ISAC provide design constraints on the UAV applications. To verify that a UAV behaves as required, the infrastructure enforces runtime monitoring, allowing the behavior, performance, and reputation of each UAV to be tracked with respect to constraints specified in the ISAC. When necessary, it responds with runtime, procedural, or even regulatory actions (cf. Fig. 1 for an overview of the different phases). For example, if the reputation of an individual UAV drops below an acceptable level, perhaps because the UAV fails to consistently report its current position, then the infrastructure could increase the minimum separation distance from other UAVs in the area, and/or could instruct the UAV to leave the area or even deny it future entry. In the evaluation we focus on non-malicious behavior and on violations caused by malfunctions or operator error, rather than on malicious behavior such as deliberate violations of restricted areas, collisions or transmission of incorrect monitoring data.

Our approach is currently implemented in a stand-alone environment; however, the use of Interlocking Safety Cases, which is the primary contribution of this paper, could be implemented within other UAV traffic management systems such as NASA’s UTM. Our approach provides several key contributions to the use of safety assurance cases for enhancing safe shared airspaces:

1) It provides an environment in which the ISAC can be rigorously constructed using commonly adopted safety analysis techniques such as Failure Mode Effect Criticality Analysis (FMECA) [12] and consequently provides safety guidance to software developers constructing UAV applications.

2) It achieves this by enriching the SAC with annotations defining interlock points where potentially untrusted pSACs for individual UAVs can interface with a trusted and assured ISAC to provide static or runtime checkable constraints.

3) Runtime monitoring is established dynamically as each UAV approaches the controlled airspace, and the monitoring is maintained throughout the UAV’s presence within the airspace.

4) Finally, we augment an existing reputation model to work effectively in the UAV context by dynamically updating a UAV’s reputation based on its compliance with ISAC constraints.

Evaluating system-level solutions is challenging as their performance is impacted by both the software and systems behavior of the application [13], and the real test of their effectiveness is only fully realized over time as they are adopted by users and deployed in the targeted environments. However, as an initial evaluation, we conducted a series of experiments based upon our experiences with two urban UAV community projects in which we met with stakeholders, elicited requirements, and explored potential solutions. One of these projects (river search and rescue) has led to an implemented system with initial features currently under test in collaboration with the South Bend fire department on the St. Joseph River.

We evaluate our approach in three primary ways: first through a qualitative analysis of the process of developing the infrastructure for a shared airspace, its associated ISAC, and representative pSACs for the two community projects; second through conducting extensive high-fidelity simulations to evaluate the effectiveness of our monitoring and reputation models; and finally through conducting field tests with physical UAVs.

We seek to explore the following research questions:

- RQ1: Is our safety approach, based on Interlocking Safety Cases, practical for use and sufficiently expressive to handle the scope and diversity of safety concerns in scenarios where UAVs are deployed in a shared urban airspace?
- RQ2: Is our monitoring approach, driven by assumptions in the ISAC and corresponding constraints in the pSAC, capable of detecting UAV safety violations, and dynamically building a reputation model that accurately portrays the behavior of UAVs flying in a shared airspace?

In the remainder of the paper, Section 2 introduces safety...
assurance cases and motivates our work. Section 3 presents our approach for safe UAV applications and introduces the novel concept of Interlocking Safety Cases. Sections 4 and 5 describe our use of runtime monitoring and reputation modeling to evaluate UAV compliance to ISAC constraints at runtime. Section 6 reports our experimental evaluation. In Section 7 we discuss our results and findings and report threats to validity. Finally, in Sections 8 and 9 we discuss related work and conclusions.

2 BACKGROUND

Our proposed work incorporates prior findings from two key research areas. These are the use of safety assurance cases to analyze software intensive systems, and runtime monitoring and verification. In this section, we provide a brief introduction to each of these areas, and then describe the context of our proposed system using a problem frame which provides a well-established approach for describing requirements of a system-to-be [14].

2.1 Safety Assurance Cases

A safety assurance case, such as the one depicted in Fig. 2, provides a semi-formal argument to justify claims that the system will operate safely in its intended environment. There are many different ways to represent safety cases, such as CAE (Claims, Arguments, and Evidence) or ARM (Argument Metamodel) [15]. In our work we adopted the Goal Structuring Notation (GSN) [11], in which arguments are constructed from goals, strategies, solutions, assumptions, and context, due to its widespread usage. Goals represent assertions that the system satisfies key requirements and properties within a specific context of use. For example, goal G1.1 asserts that "GPS is accurate within 5 cms," which contributes to the higher level assertion that the collision avoidance algorithm (G1) provides correct instructions for avoiding in-air collisions. Strategies specify the inference rules from goals to their sub-goals, both of which may depend on assumptions that must be shown to be valid. Arguments link claims to evidence in order to demonstrate that the claims are met in the delivered system. These arguments can be boolean, probabilistic (e.g., based on simulations), or qualitative (e.g., regulations or manufacturer’s specifications).

System-level SACs that address all safety-related hazards can be challenging and time-consuming to create due to the complexity of most safety-critical domains, the interdependencies of software, hardware, process, and operating environments, and the fact that alternate argument structures and decompositions are possible [16]. They are generally created in a complex process involving safety-analysis techniques such as Hazard Analysis [17], Software Fault Tree Analysis [18], Software Failure Modes, Effects and Criticality Analysis [19], [20], and safety evidence including design and code verification and reviews, and safety audits. The goal of the safety analysis processes is to identify hazardous states and their mitigating, safety-related requirements [11], [21], [22].

SACs have already been introduced into the UAV domain. For example, the US Federal Aviation Administration’s Flight Standards Information Management System
This enables assurances for the safety case similar to those offered by assume-guarantee reasoning at the component level [31]. However, our “trust but monitor” approach is applied to safety case elements at runtime, rather than to system components, and is actionable in that it specifies the evidence needed.

2.2 Runtime Monitoring and Verification for CPS

As software systems are subject to continuous change and on-going evolution during their lifetime [32] it is necessary to observe and verify their behavior during operation. This is also true with respect to a UAV infrastructure and the diverse UAVs, ground control stations, and other systems that must collaborate within the shared airspace. Within the context of a system of systems (SoS), where many different, independently developed components are required to work together to fulfill a common goal, monitoring their properties at runtime has been demonstrated as a viable solution for detecting deviations from desired behavior [33]. Similarly, for cyber-physical systems (CPS) that exhibit SoS characteristics, runtime monitoring and/or runtime verification enable observations to ensure that interconnected hardware and software components adhere to specified constraints or contracts. Although runtime monitoring and verification are closely connected and sometimes used interchangeably, they have certain differences. While verification relies on formal models, automata, or various logic notations and can be found in many military applications of CPS [34], [35], self-adaptive systems, or in the domain of autonomous vehicles [36], [37], [38], runtime monitoring describing properties of a system, for example via constraints, or domain-specific languages that can be checked at runtime [39].

As previously explained, given the current UAV environment, the typical skills of commercial developers working in the space, the cost of developing formal models, and the limited opportunities for reuse across diverse UAV environments, we focus our solution on runtime monitoring rather than formal verification. Our approach does not preclude the use of formal verification methods in individual UAV applications, or within the airspace infrastructure; however, it leverages runtime monitoring, rather than formal verification, to check that each UAV meets its expected runtime behavior and constraints.

2.3 Problem Description

We position our proposed solution within the well-understood context of problem frames [14], [30], which are used to structure, clarify, and present the requirements of the system-to-be within its system context. Strunk and Knight [40] previously showed how problem frames and assurance cases can be usefully combined to argue satisfaction of the framed problem and illustrated this with an example of a runway safety monitor. Fig. 3 provides our problem frame decomposition of the urban infrastructure and the prototype we developed to detect discrepancies between the infrastructure safety assurance case and the safety assurance cases of the UAVs seeking entry to, or flying within, that airspace.

The infrastructure provides information to the airspace controllers, both human and automated, and is therefore an instance of Jackson’s information display frame, which captures a pattern for the solution of the problem [30]. It obtains, processes and presents the information about the physical world, in this case focusing on the UAVs that populate the airspace at a point in time, as this information is needed by the urban infrastructure authorities to maintain a safe airspace. If the monitoring detects information about the physical world that breaks assumptions in the infrastructure’s safety assurance case, it can alert the airspace authorities.

As depicted in Fig. 3, the solution includes contextual models for the infrastructure’s violation-alerting machine, namely the models of the regional airspace in which the UAV are monitored, the ground control stations responsible for the monitoring, the monitored air traffic data, the UAVs’ safety assurance cases (pSACs), the infrastructure’s safety assurance case (ISAC), and the integrated displays provided by the infrastructure itself. These contextual elements are the sources of the assumptions and constraints whose satisfaction the safety assurance case seeks to ensure.

The top element of the decomposed problem frame in Fig. 3 is the model of the pSAC that each UAV must have on file with the airspace authority, while the bottom element is the model of the ISAC, whose validity the infrastructure must monitor. The other elements in Fig. 3 provide real-time data (traffic, airspace, ground control, displays) needed to monitor the airspace safety case’s validity and to issue alerts if its validity is placed at risk. The solution that we propose to achieve this is to interlock the safety case models (the top and bottom elements in Fig. 3) in order to continuously monitor the infrastructure’s safety.

3 INTERLOCKING SAFETY ASSURANCE CASES

Our solution splits the overall SAC into two types of interlocking parts: the Infrastructure Safety Assurance Case (ISAC) focuses on achieving infrastructure level safety goals, such as aiding in the prevention of UAV collisions by sending directives to UAVs in the controlled airspace. There is only one ISAC for each UAV ecosystem. In addition, Pluggable...
Safety Assurance Cases (pSACs) represent the second type of SAC, and are used, in our scenario, to specify the safe operation of individual UAVs.

More precisely, an ISAC specifies assumptions on the behavior and properties of each UAV in its airspace, while the pSAC provides claims and associated arguments that such assumptions are met. Each individual UAV therefore possesses its own pSAC, which interlocks (at runtime) with the ISAC when it enters the controlled airspace. The interplay between the ISAC and its interlocking pSACs is depicted in Fig. 4. We employ the Goal Structuring Notation for both types of safety assurance case and add additional annotations, properties, and constraints to the nodes of both the ISAC and pSAC. By plugging sub-trees of the pSAC into the ISAC’s respective interlock points, the infrastructure can ascertain whether a UAV provides sufficient capabilities to operate within the controlled area and can also monitor safety-related runtime properties.

Fig. 4: Elements and properties of the ISAC and pSAC. An interlock point connects measurable constraints defined in the ISAC with SAC fragments defined in the pSAC.

### 3.1 Infrastructure Safety Assurance Case (ISAC)

The ISAC is created following common SAC construction processes [11], [41] with extensions to facilitate interlocking with the pSAC. To achieve this, **interlock points** are established between UAV specific goals in the ISAC and solutions that the pSAC provides. The connecting items are constraints specified in the ISAC with the assumption that the pSAC can, and will, meet those assumptions. This is illustrated in the example depicted in Fig. 5. The ISAC asserts in goal G1 that a UAV flying in the managed airspace will not collide with fixed ground-based obstacles. This is a seemingly audacious goal as UAVs are independently operated and controlled. However, the goal is broken into three sub-goals, each of which sets an assumption, realized as an ISAC constraint, on the behavior and/or properties of each UAV. Sufficiently satisfying the ISAC goal (G1) is therefore dependent upon all UAVs operating within the area. Our approach includes constraints which are associated with pSAC solutions in three distinct ways: @entry, @monitor, and @psac. The first two represent assumptions associated with monitorable UAV properties, while the third represents assumptions that are too complex to be fully monitored and therefore require an extension of the SAC. The risk of collision with a fixed ground based object is sufficiently mitigated if all three sub-goals are addressed, meaning that the system is acceptably safe.

@entry represents a property that is monitored one time upon UAV entry into the space. For example, G1.2 (cf. Fig. 5(a)) claims that the infrastructure provides a map documenting the coordinates and height of all FGBs in the managed area. The infrastructure checks that the associated Constraint CS2, specifying that each UAV holds a copy of the current infrastructure map documenting coordinates of all fixed ground-based obstacles, is true upon UAV entry into the controlled area. Additional @entry properties may include hardware related checks, such as the maximum size and weight of the UAV, speed and maneuverability capabilities, payload data (such as supplies, camera equipment, etc.) or checks for software capabilities such as onboard obstacle avoidance or accuracy of terrain detection (i.e., reliance upon a map API versus potentially far more accurate sensor-based capabilities).

To support runtime checking, the @entry property is transformed into a (checkable) constraint. Any sufficiently expressive constraint language (e.g., the OCL [42]) can be used to specify and subsequently check conformance to the constraints at runtime. For example, the constraint associated with the @entry property defined in CS2 could be specified as follows:

\[
\text{uav\_mapVersion} \rightarrow \text{inf\_requiredMapVersion}
\]

Upon entry, the UAV presents its current map version, which is then compared to the one required by the infrastructure.

@monitor represents a property that is monitored continuously, or on demand, whilst the UAV is in the controlled area. This is illustrated through goal G1.1 which asserts that each UAV knows the altitude of the terrain in its current vicinity. In this example, the constraint CS1 captures one aspect of this claim that the UAV knows the current distance from the ground and reports it every x seconds. The monitorable property is here transformed into two constraints:

\[
\text{uav\_distToGround}.\text{timestamp} < \text{uav\_lastDistToGround}.\text{timestamp} + \text{inf\_distToGroundPeriod}
\]

\[
\text{abs} \left( \text{uav\_distToGround} - \text{inf\_sensed\_uav\_distToGround} \right) < \text{inf\_acceptAltitudeError}
\]

where uav_distToGround is the distance reported by the UAV, inf_sensed_uav_distToGround is the distance from the ground computed by the infrastructure using its own sensors, and inf_acceptAltitudeError is the degree of acceptable error between the two values. It is worth noting that we do not specify how the UAV should detect the distance from the ground, for example, this could be achieved using a distance sensor (e.g., sonar, radar, or LiDAR) or by use of accurate GPS and querying an accurate map. The second constraint is an example in which the infrastructure not only requests a property from the UAV but uses its own environmental sensors to externally vet the value for correctness. We revisit the runtime monitoring annotations, @entry and @monitor, in Section 4 and describe how such values feed into our UAV reputation model.

Finally, @psac tags represent assumptions which cannot fully be represented by a simple set of monitorable...
properties. This could, for example, be due to alternative technologies that might be adopted to achieve the same goal, in which case it is not desirable to monitor properties that are internal or unique to specific design solutions. We provide an example through goal G1.3, which asserts that each UAV will avoid fixed ground-based obstacles. This assumption addresses the need for the UAV to be aware of its surroundings and to provide on-board sensors (such as LiDAR or visual obstacle avoidance) to detect and avoid collisions with structures such as building or other ground-based infrastructure. The ISAC seeks assurance that the UAV solution adequately addresses potential causes of collisions and that the UAV will not collide with a fixed object. In this case the ISAC requires each UAV to submit their custom SAC prior to entry into the airspace. This pSAC may be inspected by infrastructure operators as deemed necessary.

3.2 Pluggable Safety Assurance Case (pSAC)

Each assumption made by the ISAC that is associated with an @monitor, @entry, or @psac assumption must be reflected in the pSAC. This is illustrated in Fig. 5(b) for a specific UAV instance. Responding to @monitor and @entry constraints in the pSAC is rather straightforward and typically requires a claim that the UAV has met the ISAC generated constraint, supported by associated evidence. This is illustrated by claim G2.1 and its associated solution Sn1 documenting the use of test cases. A mapping is then created between the concrete UAV and variables used in ISAC constraints. This could include simple name mappings, unit conversions (e.g., imperial to metric measures), or even more complex computations to ensure that the reported data is in the format required by the ISAC. For example in Fig. 5(b), the UAV internal variable position.z in G2.1 is mapped to its corresponding variable defined in the ISAC constraint (i.e., uav_distToGround). This mapping allows the infrastructure to monitor behavior of the UAV as required by the pSAC.

Responding to @psac requests is more challenging. The UAV developer must provide a full argument demonstrating how the ISAC assumption has been met. For example, the subtree associated with claim G2.2 in Fig 5(b) addresses the ISAC’s constraint that each UAV will never collide with a FGB obstacle. The pSAC makes three sub-claims that it plans routes to avoid obstacles, leverages the current map of fixed ground-based objects provided by the ISAC, and utilizes the manufacturer’s onboard sensor and avoid technologies. While our approach currently does not dynamically analyze pluggable @psac arguments, the pSAC must be provided prior to entry into the controlled airspace. In case of incident (i.e., accident, malfunction, or behavior that deviates from expectations), the full pSAC is available for further evaluation and, if necessary, can be used to hold the UAV operator accountable.

One of the benefits of our approach is that the use of our pSAC across multiple systems allows us to create templates and wizards to aid UAV developers and manufacturers through the process of constructing a pSAC. The wizards simplify the process of complying to safety standards, providing guidance through the software and systems development process. For example, in the requirements phase, using the wizard would help a UAV developer to understand and specify hard constraints on their system. These constraints will guide critical aspects of the design, implementation, and testing phases of the software system [43], [44]. Reuse is applicable at two levels (1) through a knowledge base of potential claims and sub-claims associated with common behaviors that are likely to occur across many ISACs, and (2) by matching capabilities of popular UAV hardware to these claims. Sub-claims G2.2.1 and G2.2.2 could be suggested...
3.3 Creating Interlocking Safety Cases

The process for defining and interlocking the ISAC and pSACs is illustrated in Fig. 6. The first step involves creating a domain-level ISAC which includes goals, strategies, solutions, and constraints that are expected to be relevant across many different application spaces. In addition to constructing the core SAC, the process also includes specifying annotations (i.e., @entry, @monitor, @psac), constraints, and global properties. Steps 1-4 in Fig. 6 are executed by airspace operators in order to customize the domain ISAC for their own usage context.

In step 1, new safety hazards may be identified and added, or existing ones not relevant to the current airspace may be removed or modified. For example, our original domain model was created for a relatively flat area of the river, and we did not consider hilly terrain. However, different goals are needed for flying in spaces with such terrain. Customization may involve adding, removing, or modifying entire sub-trees, or individual nodes related to safety claims, sub-claims, specific properties or annotations used to create interlock points with pSACs. These modifications may be merged back into the domain model, to make them available for future airspaces. While ISAC evolution is outside the scope of this paper, we envision modeling the domain-level ISAC as a product line, and allowing its evolution as new, potentially reusable, features are identified at the individual ISAC level and promoted to the domain level [45]. Both the domain level, and the project-specific ISAC can be modeled in any SAC tool with capabilities for exporting to XML, JSON, or other formats.

Using the exported ISAC from the previous step, in step 2, we automatically generate an ISAC template for the targeted UAV airspace. If necessary, in step 3 we can manually customize and adapt specific properties in the template such as the maximum allowed wingspan, weight restrictions, or even monitoring rates. Finally, in step 4, we generate the pSAC template which then serves as the foundation for each individual operator to create a pSAC for each UAV that will fly in the airspace. This pSAC template contains information, needed by the UAV developer, such as constraints to be satisfied and variables to be exposed at specific interlock points.

Steps 5-6 represent the responsibilities of individual UAV operators. During the development process, in step 5, the UAV operators instantiate the pSAC template and configure it according to capabilities and properties of their own UAV. Tools can provide semi-automated support for aiding users in the configuration process – for example, by retrieving information about UAV capabilities from a database containing basic information (e.g., maximum speed or payload capabilities) on UAV models and versions. The ISAC constraints expose a set of monitored variables needed to evaluate the UAV, which must be provided by the system represented by the pSAC. In step 6, the pSAC certificate is generated and stored at the UAV’s ground control station or on-board the UAV.

Steps 7 and 8 represent monitoring behavior, which is described in Section 4. Finally, in step 9 monitoring information is used at runtime to identify any violated ISAC assumptions so that responsive measures can be taken. These may include notifying the UAV operator of a compliance problem, temporarily increasing separation distance around the UAV, or even notifying authorities in case of severe safety infringements.

The process is designed to provide steps and tasks which can be fully supported by external tools with software support for automation. Our implementation is described in Section 5. The aim is to facilitate development of safely operating UAVs and UAV applications by enabling developers to create pSACs without detailed knowledge of the safety requirements of the infrastructure.

Like any other system, the infrastructure, environment, regulations, and UAVs themselves are all subject to change, and new types of faults may be observed over time. As a
result, it may be necessary to add new goals to the ISAC leading to additional constraints to be satisfied by the UAVs. Following our described process this can be addressed by regenerating the respective ISAC and pSAC templates and re-deploying them to the monitoring system.

4 Runtime Monitoring with Safety Assurance Cases

Diverse monitoring approaches for various kinds of systems and purposes have been proposed including requirements monitoring and performance monitoring [46], however very few of them address monitoring of safety-critical properties or safety assurance cases in particular [47], [48]. Efforts have been made to introduce runtime safety assurance for cyber-physical systems, especially in the automotive domain [49], [50]. While the underlying monitoring techniques we use here have been well described in other publications [33], [39], the novelty of our approach arises from a customized monitoring solution that supports the interlocks established between the ISAC and specific pSAC instances. These interlocks are established upon the UAV entry into the airspace and enable UAV behavior and properties to be monitored as previously described. In addition to dynamically establishing runtime monitoring, we propose and integrate a reputation-based approach where each UAV can gain or lose reputation based on constraint evaluation results.

The three monitoring steps were depicted in Fig. 1. The first step executes when the UAV requests access to the controlled airspace and the UAV (or its respective ground control station) presents its pSAC to the infrastructure (V1). The infrastructure checks all the pSAC’s @entry claims against the ISAC constraints. If any @entry constraint is not present or not satisfied in the pSAC, then the UAV is denied entry. In the same step it evaluates whether the @monitor points have been established correctly, meaning that the pSAC has provided mappings between local UAV properties and the pSAC properties used in the constraints. If both checks succeed, the UAV is granted permission to enter the airspace. While the UAV is operating in the controlled airspace it is required to provide monitoring messages which the infrastructure uses to evaluate @monitor constraints (V2).

Monitoring requires an interlock to be established between the UAV (or its respective ground station) and the runtime infrastructure that manages the ISAC, so that monitored data can be sent from the UAV to the infrastructure and monitoring directives sent from the infrastructure to the UAV. In this paper, we focus on the creation of the ISAC and pSACs, and the end-to-end concepts of monitoring, rather than providing an in-depth description of the monitoring implementation. However, we briefly describe the prototype we developed and used for our experimental evaluation in Section 5.

4.1 Reputation-based monitoring

The frequency at which UAVs are required to provide monitoring data regarding their current state, speed, gps accuracy, etc., is determined dynamically based on each UAV’s reputation. We reviewed several existing reputation models [51], [52], [53], but ultimately chose a popular class of model [54], [55] that is based on the beta distribution and is largely influenced by the work of Josang and Ismail [56]. We selected this model over more ad-hoc approaches as it is statistically sound and performs well in uncertain, dynamic environments such as the UAV airspace. The reputation of a single UAV is based on the set of assumptions $A = \{ A_i \mid i \in \mathbb{N} \}$ that have been defined as @monitor points by the ISAC. Results of constraint evaluations (associated with these assumptions) are stored as feedback tuples $A_i = \{ (r_{i1}, s_{i1}), \ldots, (r_{im}, s_{im}) \}$ where $r_{ij}$ represents the degree of positive feedback, $s_{ij}$ represents the degree of negative feedback, and $m$ is the number of times that $A_i$ has been evaluated at a current point in time [56]. This data structure can be thought of as a matrix, where each row is an assumption and each column is a constraint evaluation during monitoring. Thus, the number of columns grows as monitoring proceeds over time.

Feedback tuples are recorded separately for each monitorable assumption, i.e., the reputation of a UAV is defined as the aggregation of its assumption reputations. This has two benefits: first, the system can adapt its behavior on the assumption level rather than the UAV level, and second, individual assumptions can be assigned more or less importance by defining a constant weight for each monitorable assumption in the ISAC (cf. Fig. 5). The weight of an assumption is used to describe its “importance”, and in turn reflects the criticality of the violated assumption. For example, a UAV entering a no-fly zone is a more critical violation than a UAV slightly exceeding the speed limit. Initial weights are defined in the ISAC template which then need to be adapted for a specific instance of the ISAC, i.e., a certain scenario or area of operation (cf. Section 3.3).

Intuitively, more recent feedback is a better indicator of future performance. This is due to the fact that UAV behavior is not fixed but rather fluctuates over time as environmental factors, health of the UAV, and stakeholder interests change. We can account for changes in UAV behavior by introducing a forgetting factor $\lambda$ which can be adjusted based on expected behavior volatility. For each assumption $A_i$, we define the positive and negative feedback as [56]:

\[
\hat{r}_i = \sum_{j=1}^{m} A_{ij} \lambda^{m-j} \quad \text{and} \quad \hat{s}_i = \sum_{j=1}^{m} A_{ij} \lambda^{m-j},
\]

where $\lambda \in [0, 1]$. Then, the reputation of a UAV can be defined as [56]:

\[
R_{uav} = \frac{1 + \sum_{i=1}^{n} \hat{r}_i}{2 + \sum_{i=1}^{n} \hat{r}_i + \hat{s}_i}.
\]

The reputation $R_{uav}$ at a point in time can be thought of as the likelihood that the UAV will uphold its claims. Initially $R_{uav}$ is equal to 0.5, i.e., there is a 50% chance that a violation will occur. As time progresses, the reputation $R_{uav}$ will fluctuate between 0 and 1, depending upon the results of monitoring. We use the reputation of each UAV to specify a monitoring period. The rationale is that less reputable UAVs should be monitored more frequently (V3). This is of practical importance as monitoring requires significant bandwidth. The monitoring period for a UAV is defined as:

\[
T_{uav} = R_{uav}^\rho \ast (\beta - \alpha) + \alpha,
\]
where $\alpha$ and $\beta$ are the lower and upper bounds respectively, and $\rho$ is a constant used to control the rapidity of change in the monitoring period for a UAV. For example, setting $\alpha = 5$ and $\beta = 60$ means that the UAV has to provide monitoring information at least every five seconds (in case its reputation is low) and at most every 60 seconds (in case it is high). The period ($T_{\text{uav}}$), between the upper and lower bounds ($\alpha$ and $\beta$), at a certain point in time is then calculated based on the current reputation of the UAV.

## 5 Prototype Implementation

To support experimentation, and to serve as a beginning-to-end proof of concept for our complete process, we created a prototype implementation encompassing all of the steps outlined in Fig. 6. Here we briefly report on the details of the implementation, focusing on the monitoring and reputation model used in our simulations (cf. Section 6).

We created SACs using the CertWare editor [57], an Eclipse-based open source tool for building safety assurance cases. In order to customize and create the respective ISAC and pSAC templates (cf. steps 2 and 4) we extended the editor with a parser which we used to transform the SAC into an XML-based template including annotations, properties, and constraints associated with the interlock points. This step is fully automated and generates a full ISAC based on a standard safety assurance case. The resulting ISAC can easily be configured and manually adapted (i.e., modifying its overall structure, properties, and annotations) in CertWare or in the generated XML file. We further automatically generated the respective pSAC template with the interlock points prescribed by the ISAC. The resulting pSAC template serves as an input mechanism for our configuration tool allowing UAV developers to enter mappings and properties for @entry and @monitor claims, and to initialize new SAC snippets to address @psac annotations. Finally, the configuration tool was used to generate a JSON file which served as the pSAC for that particular UAV. It can be presented to the infrastructure by the ground control station (cf. V1, Fig. 1) when the UAV requests entry to the infrastructure’s airspace (cf. steps 3 and 4).

To support the second part of our experiments we implemented all components necessary for performing simulations of UAVs flying in a specified airspace as well as the support structure needed for experimentation with physical UAVs in field testing. This included two independently developed Python-based ground control stations capable of bidirectional communication with both physical and simulated UAVs. The ground control stations support diverse UAV tasks such as executing search patterns or flying to waypoints. We also implemented the urban infrastructure, which is capable of observing and coordinating UAVs in the airspace, as well as the reputation and monitoring model for monitoring UAVs at runtime.

Both ground control stations can handle multiple UAVs that each present their respective pSAC to the infrastructure when either a simulated or physical UAV is about to enter the airspace. The implementation of the infrastructure itself relays information to the monitoring component, which performs the previously described @entry checks and then grants or denies the UAV entry into the airspace. A socket-based connection is then established between the ground station and the control infrastructure for relaying information from UAVs to the infrastructure and receiving commands as to the frequency at which such information must be provided. As long as the ground station has UAVs operating in the airspace it provides respective monitoring messages at the variable rate requested by the infrastructure.

Both the monitoring component and the infrastructure are implemented in Java, and we use the Oracle Nashorn JavaScript Engine [58] to dynamically generate and evaluate constraints specified in the ISAC. We define our constraints as lambda-expressions which are translated (literally) on-the-fly into JavaScript functions and executed.

## 6 Experiments

Our experimental evaluation was designed to explore the following research questions:

RQ1: Is our safety approach, based on Interlocking Safety Cases, practical for use and sufficiently expressive to handle the scope and diversity of safety concerns in scenarios where UAVs are deployed in a shared urban airspace?

RQ2: Is our monitoring approach, driven by assumptions in the ISAC and corresponding constraints in the pSAC, capable of detecting UAV safety violations and dynamically building a reputation model that accurately portrays the behavior of UAVs flying in a shared airspace?

### 6.1 RQ1: Applicability to real-world scenarios

We refine RQ1 into two sub-questions:

RQ1.1: How much effort was required to construct the ISAC and pSAC for a set of safety goals and was this effort significantly reduced through reuse of the ISAC and pSAC in subsequent scenarios?

RQ1.2: What are the challenges associated with constructing an ISAC and pSAC for a specific airspace/infrastructure and its constituent UAVs?

**Research Design:** To explore the applicability of our ISAC and pSAC approach in real-world scenarios, we created and customized two different ISACs based on community projects in which we were engaged: one for a toll-road (TR) accident surveillance scenario and one for a city infrastructure (CI) which includes search-and-rescue on the city’s river, surveillance of building fires, as well as commercial and hobbyists’ uses [59]. Due to the fact that analyzing all hazards and mitigations for UAV flights in an urban area is a time-consuming task, we focused our efforts on addressing four important safety goals that the infrastructure would support: (1) prevention of UAV to UAV collisions, (2) prevention of UAV collisions with physical objects on the ground, (3) ability for a UAV to find safe landing sites in case of an emergency, and finally (4) ensuring UAV compliance with regulations related to no-fly zones, maximum payloads, etc.

In preparation for developing the ISAC we met with members of the South Bend fire department and city administrators and had discussions with the toll-road stakeholders to identify their scenarios of use. We then hosted our own internal workshop at which we performed an initial hazard analysis, identified contributing faults and their potential
mitigations, and worked collaboratively to construct a first-cut ISAC for an urban UAV ecosystem.

Workshop participants included a licensed UAV pilot with many years of experience flying UAV missions for scientific data-collection purposes, a safety expert with over ten years of industrial experience in the safety-critical domain, three software and systems engineers with experience developing software for UAVs, and an HCI expert. As a result of the workshop we developed a domain-level ISAC with annotations depicting where interlock points might be established. We then systematically refined the ISAC through additional informal whiteboard meetings and online discussions to elaborate the ISACs and pSACs by identifying and including a more extensive set of interlocks, global property definitions, and constraints.

We next customized the ISAC to reflect specific requirements and safety hazards associated with each UAV urban infrastructure environment. We provide the complete ISAC, annotated to show which claims and associated constraints are applicable for each environment, in the supplementary material. As explained in our implementation description, ISAC templates and pSAC templates were both generated automatically from the customized ISAC. We then used the pSAC template to configure two different instances for the 3DR IRIS+ quadcopter and two instances for the S900-folding wing airframe hexcopter, both with and without RTK (where RTK provides higher precision GPS) (cf. Fig. 7(c)). Focusing on the four safety goals we explored 12 safety arguments, 27 associated sub-goals, and 20 assumptions (5 @entry, 10 @monitor, and 5 @psac). While these goals are clearly insufficient for fully achieving safety, they established a non-trivial experimental environment.

**Results:** To address RQ1.1 we investigated the time and effort required to develop ISACs and pSACs for the four safety goals. It took our group of stakeholders approximately 80 person hours to produce an initial SAC addressing a limited selection of potential hazards associated with urban use of UAVs for the CI scenario. While determining how much effort is “reasonable” is highly subjective and dependent upon the domain and its criticality level, this amount of effort, while not trivial, did not seem to be excessive. Rather than detracting from our proposed solution, the time investment reinforced the need for a reusable and generic ISAC as a starting point for other engineers building urban infrastructures. The majority of human effort went into creating the initial SAC, while customizing it for use in a specific setting was much easier and less time-consuming, taking approximately four hours for the toll-road scenario. During the customization process we identified additional hazards associated with hilly terrain and curvy flight paths that followed the road without flying over the tops of vehicles. We were able to accommodate the resulting mitigations by extending and evolving the domain level ISAC. Based on our experience, we report that creating the ISAC and pSAC for our second, closely related TR domain, took only 5% of the time it took to create the initial CI version which was a significant reduction over the effort required to develop the initial ISAC.

To address RQ1.2 we discuss our experiences of creating the ISAC and pSAC, which reinforced known challenges of creating SACs including the lack of a clearly defined, reusable process guide. These challenges have been documented by numerous authors [60], [61], [62], [63], [64], [65]. Our primary challenge was finding the right decomposition of high-level goals. In the case of our ISAC and pSAC this decomposition needed to be rather extensive so that the lowest level of sub-goals would align with the low-level UAV characteristics from which respective interlock points and constraints are defined. We observed clear trade-offs between abstraction and detail in designing an appropriate goal hierarchy.

A bottom-up approach resulted in the definition of goals with overly narrow scope, and we ended up with several sub-goals that referred to the same (or similar) constraints and shared the same monitorable properties. This added unnecessary complexity to the overall SAC and introduced redundancy (clones) across different parts of the SAC. On the other hand, we found that while a top-down approach was effective for refining infrastructure goals, it failed to fully consider the hardware characteristics of UAVs operating in the airspace and therefore failed to provide adequate interlock points for connecting to monitorable UAV properties (i.e., constraints). By combining the top-down decomposition with a bottom-up approach that investigated UAV capabilities and then systematically linked them to infrastructure goals, we were able to establish the respective

Fig. 7: Scenarios used for the evaluation. In the Toll-Road surveillance, UAVs will be used by toll-road authorities, emergency services, and news outlets to survey accident areas. The City Infrastructure is designed to control access to, and coordinate the use of UAVs in an urban airspace – including search-and-rescue on the city river, goods delivery, and hobbyists.
interlock points and bridge the gap between the ISAC and pSAC.

Once the @entry and @monitor points were identified and specified, we found them relatively easily to configure for individual pSACs. Using open source UAV hardware and software our main effort involved studying documentation and publicly available material to identify specifications and parameters. The effort might be considerably higher for commercial UAVs with closed-source environments; however, in the future, vendors could provide certificates or prefigured pSACs for certain types or models.

On the other hand, satisfying @psac points by providing a custom pSAC composed of goals, strategies, contexts, and solutions required significantly more effort and would likely require support from the respective vendors. Wherever feasible, our approach favors the use of @entry and @monitor interlocks over the use of @psac points. In summary, for RQ1.2 we found that while constructing the ISAC required domain expertise and took several design iterations, the resulting interlocking approach was practical and sufficiently expressive to create safety cases and their interlock points for two urban, real-world scenarios.

6.2 RQ2: Runtime Monitoring of ISAC/pSACs

For answering the second research question we leveraged the ISACs and pSACs created for the toll-road and city ecosystems to evaluate whether our proposed runtime monitoring and reputation model was capable of detecting violations of assumptions defined in the ISAC. The evaluation was conducted in two phases – first as a series of simulations using the high-fidelity ArduPilot Software-in-the-loop (SITL) simulator [66], and then as an outdoor field test with physical UAVs.

Our main goal was to determine whether the approach is suitable for detecting unexpected and undesired UAV behavior and to better understand the extent to which assigning different weights to different assumptions affected the reputation and monitoring period of the UAV. The simulations covered a range of scenarios (described below), each with increasing numbers of UAVs. Furthermore, our evaluation aimed to show that multiple pSACs (i.e., representing all the UAVs operating in the area) could be plugged into a single ISAC.

<table>
<thead>
<tr>
<th>ISAC</th>
<th>Scenario</th>
<th>Ass.</th>
<th>UAV</th>
<th>[#(a/b/c)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toll-Road Surveillance</td>
<td>TR-SC1</td>
<td>5</td>
<td>20</td>
<td>(4/4/12)</td>
</tr>
<tr>
<td></td>
<td>TR-SC2</td>
<td>6</td>
<td>40</td>
<td>(8/8/24)</td>
</tr>
<tr>
<td></td>
<td>TR-SC3</td>
<td></td>
<td>40</td>
<td>(8/8/24)</td>
</tr>
<tr>
<td>City Infrastructure</td>
<td>Ci-SC1</td>
<td>5</td>
<td>50</td>
<td>(20/16/14)</td>
</tr>
<tr>
<td></td>
<td>Ci-SC2</td>
<td>6</td>
<td>100</td>
<td>(40/16/44)</td>
</tr>
<tr>
<td></td>
<td>Ci-SC3</td>
<td></td>
<td>100</td>
<td>(40/16/44)</td>
</tr>
<tr>
<td>Physical UAV</td>
<td>Flying field</td>
<td>5</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

For each ISAC we anticipated three different scenarios: The first scenario (SC1) was used to establish a fault-free baseline in which all UAVs adhered to their claims during the simulation period. In scenario two (SC2) we injected faults for a predefined period of time (i.e., 10 minutes) into a random 20% sample of the UAVs. We observed whether the infrastructure was able to detect and react to faulty behavior, i.e., by decreasing the reputation and the monitoring period for the UAVs and providing a notification when a UAV fell below a certain critical level of reputation. This scenario simulates a temporary malfunction of hardware (e.g., GPS) or a UAV violating no-fly zones for a certain amount of time. The third scenario (SC3) investigated the effectiveness of the system in the presence of random or periodic UAV failures. We again selected 20% of the UAVs to be defective and assigned a fault probability to each defective UAV. This probability was the likelihood, at each time-step, that a fault would be injected. In order to ensure that we obtained a balanced distribution of fault probabilities we performed 20 runs of scenario SC3 for each ISAC.

We created faults associated with three common types of moving violations, namely speeding, violating zone restrictions, i.e., entering no-fly zones, and ignoring altitude restrictions (flying too low over a crowd). Our goal was not to provide full coverage of all possible violations, but to inject a set of representative errors and check them against the constraints derived from the ISAC. Each of these violations could be triggered by deliberate or accidental operator error, or as a result of faulty equipment. We do not attempt to differentiate between the two in this paper, and treat both as reputation-reducing events. Furthermore, each violation was assigned an individual weight to investigate the effects of different weights and fault probabilities on our UAV monitoring environment.

Each scenario was executed in a 30 minute simulation (an upper bound on the average length of time many small UAVs can fly). Based on the capabilities of the simulator we used five @entry and eight of the ten defined @monitor constraints (six for each ISAC– cf. Table 2). For the toll-road ISAC we used 20 UAVs flying at the same time for SC1, and 40 UAV for SC2 and SC3. For the urban infrastructure we significantly increased the number of UAVs, using 50 UAVs for SC1 and 100 UAVs for scenario SC2 and SC3. Furthermore, to create a realistic simulation, UAVs were assigned specific tasks (cf. Fig. 7). As shown in the right-hand column of Table 1, for the toll-road ISAC we included (a) UAVs owned by the toll-road authority patrolling the area; (b) emergency units performing traffic-surveillance; and (c) news-stations video-streaming live from an accident site. For the city infrastructure we generated flight-routes for (a) eCommerce deliveries; (b) search-and-rescue UAV doing a coordinated search in a defined area; and (c) hobbies flying random routes in randomly generated areas (e.g., a park or flying field).

All experiments were run on desktop machines with Intel(R) Xeon(TM) CPU, 64GB RAM running Ubuntu 14.04 and using the latest version of the ArduPilot SITL simulator. For all three scenarios we were interested in the time required for performing the initial (on entry) validation, the number of constraint evaluations that were performed and the mean evaluation time for each constraint. Additionally, for SC2 and SC3 we were interested in how different weights and different fault probabilities would be reflected in our monitoring and
trust model. Table 2 provides an overview of the constraints used for the different ISACs.

We recorded information for each scenario regarding the number of constraints evaluated (and the respective time necessary for performing the evaluation), the reputation level of the UAV and the monitoring frequency.

**Results:** One of the primary goals of our evaluation was to better understand the extent to which runtime monitoring and reputation modeling could aid in achieving safety assurance claimed across the interlocking SACs. We therefore analyzed the results of the three different scenarios described above using different weights and fault probabilities. Fig. 8 provides an overview of results from four selected examples of UAVs and their respective reputation monitoring periods across the 30 minute simulation run. UAV-A represents a fault-free UAV that quickly builds reputation and, as a result, its monitoring period (how long before it is monitored again) steadily increases. Most UAVs are expected to behave in this way. In contrast, UAV-B, UAV-C, and UAV-D show the violations related to assumptions in the ISAC that were described above. UAV-B experiences a temporary hardware malfunction. It initially quickly gains reputation; however, 10 minutes into the simulation, faults are injected, and we observe a rapid decline in reputation. UAV-C and UAV-D both experience random or periodic failures and therefore exhibit an oscillating behavior in their reputations. The information gained through monitoring and through the dynamic change in the reputation of the UAV can be fed back into the ISAC to determine which safety goals and assumptions have been violated and to provide feedback to the infrastructure and UAV operators. For all six scenarios we were interested in whether our monitoring solution can cope with constraints provided by the ISAC and is able to detect undesired behavior of UAVs (cf. UAV-B, UAV-C, UAV-D). The evaluation times for a single constraint mostly remained constant, regardless of the number of UAVs in the area (cf. Fig. 8(a)) not exceeding 4 ms. We can argue that this is more than sufficient for this kind of application. More importantly, the analysis of the reputation rating (cf. Fig. 8(b) and Fig. 8(c)) shows that constraint violations immediately result in the loss of reputation rating and in turn in a decrease in the monitoring period, i.e., the UAV is more frequently monitored and its provided data is checked more frequently. Lastly, the monitoring period (how long before it is monitored again) decreases much more rapidly than it increased due to the selection of ρ in Equation 3. While in our experiments we did not enforce a ‘hard reputation rating limit’ but rather observed the change of reputation over time, additional actions could, for example, revoke permissions of a UAV as soon as it falls below a certain reputation rating.

In summary, the simulations we conducted confirmed that our approach was able to handle, and react to, different kinds of safety violations. We demonstrated that the prototype implementation of the runtime monitoring component scaled to an increasing number of UAVs in the airspace while performing constraint checks. For example, in the city infrastructure scenario with 100 UAVs operating in the airspace, over 70,000 constraint checks were performed during the 30 minute period with a mean evaluation time of approx. 2.3 ms per constraint.

For the second part of the experiments, involving physical UAVs, we performed a series of tests executing predefined flight routes using our ground control station. We flew a series of scenarios based on the same assumptions and constraints used in the city and toll-road simulations. These included, among other things, accurate positioning with and without RTK (i.e., flying to a predefined spot), adhering to no-fly zone limitations (i.e., not entering a fictitious no-fly area on the flying field) and adhering to speed limits. Our goal was to confirm that the process of pSAC entry into the space and subsequent monitoring worked effectively in a physical environment. With regard to RQ2, the experiments confirmed that the proposed approach, including entry checks, runtime monitoring and reputation modeling, worked effectively in the physical world with actual UAVs and that violations were successfully identified and mapped against their appropriate ISAC assumptions.

## 7 Discussion

Results from our experiments showed that our interlocking ISAC/pSAC approach was successfully applied across two real-world scenarios with UAV exhibiting diverse capabilities. We observed ample opportunities for customizing major parts of the domain-level ISAC for use in specific urban scenarios. The clear separation between the roles of the ISAC and the pSAC would enable the ISAC to be rigorously constructed and evaluated using traditional safety techniques. At the same time, the pSAC specified required monitorable constraints on the pSAC and its corresponding UAV which would provide clear guidance to software engineers developing their own UAV applications. For example, the constraints drive key safety-related aspects of the requirements specification and analysis, and impact software and systems design, implementation, and testing throughout the software development lifecycle.
Although the work described here is directed to urban infrastructures, the approach readily generalizes to any UAV infrastructure (e.g., wilderness areas, UAV use within large manufacturing facilities or warehouses, long hauls over water), or wherever safety is an issue. We are confident that this approach could also be useful for assuring safe operation of other remotely controlled vehicles, including unmanned underwater vehicles, and of heterogeneous fleets containing both ground and air-based autonomous vehicles. Furthermore, our approach is also suitable to be integrated within other frameworks such as NASA’s UTM or could be implemented at a smaller scale within a locally controlled airspace.

By integrating the monitoring approach and the reputation model we have provided a non-trivial proof of concept illustrating end-to-end functionality of our approach. While trust is initially established when the pSAC plugs into the ISAC, it is verified over time, through monitoring the UAV at runtime and observing the reputation rating of the UAV. Furthermore, in a broader sense, the reputation ratings of individual UAVs could trickle up to the level of UAV operators or vendors, potentially providing an incentive to adopt this approach. Our approach can not guarantee safe behavior, even when no constraints are violated. The reputation of a UAV, however, can provide an important indicator of the extent to which a UAV can be trusted and whether measures should be taken. Responses could include avoiding the immediate area surrounding a UAV that exhibits low reputation or prohibiting its access to the airspace in case of repeated or severe constraint violations.

### 7.1 Threats to validity

As with any experiment, our evaluation is subject to several threats to validity [67] falling under the categories of internal validity, external validity, and construct validity. **Internal validity** is concerned with the rigor of the experimental design. In our study, the experiments were based on the construction of ISAC and pSACs for two different application scenarios. These were created by several authors of this paper; however, the system requirements that drove the safety analysis were largely elicited from external stakeholders such as City of South Bend officials, and members of the local fire department who are actively involved in river search and rescue and emergency response. During the ISAC creation workshop, the goals, assumptions and interlock points were discussed by several experts and the safety case was iteratively created. Additionally, we provide all created ISACs as supplementary material.

**External validity** refers to how well data and theories from one setting apply to another, i.e., the generalizability of results and findings. In our experiments we considered three different dimensions: scenarios, faults, and simulations. Our results are based on analyzing two different scenarios (toll-road surveillance and an urban infrastructure scenario), and are complemented by an experiment with physical UAVs in an outdoor setting. While we cannot claim generalizability of the approach to other scenarios, our workshops and individual meetings have included experts from several different domains (i.e., safety-critical systems, UAV software engineers, UAV hardware, and end-users). More recently, we have met with additional stakeholders to discuss and develop prototypes for a broader set of emergency response

scenarios such as defibrillator delivery. Safety concerns in these later discussions have aligned with the ISAC and pSACs we previously created. We are thus confident that the created SACs are a good starting point for use in diverse urban environments and that the proposed approach is useful and applicable to a variety of different types of infrastructures and scenarios.

With respect to our simulations, we only injected faults into the UAV behavior for a selection of ISAC constraints; however, they were representative of diverse types of problems. We focused on violations caused by malfunctions or non-malicious operator error, rather than on malicious behavior such as deliberate collisions or transmission of incorrect monitoring data intended to deceive the infrastructure. We have not yet investigated deliberate security violations; however, our approach could also monitor the validity security assumptions specified in the ISAC. Finally, our experiments are based upon simulations with simplified assumptions (such as a fixed number of flight routes, and the selected constraints as described in Section 6), and on small-scale physical experiments. While our results demonstrate the viability of our approach, the ISAC needs to be extended and validated through further studies and the monitoring and reputation process evaluated in larger-scale studies with physical UAVs.

To minimize the threat of invalid data measurements in our simulation runs due to external factors such as OS tasks or interference with other applications, we performed 20 runs for scenario SC3 for each ISAC and included a warmup time before each simulation run. Furthermore, our measurements focused only on a limited number of assumptions and the constraints associated with these. However, we have demonstrated that the approach can be applied to diverse types of assumptions with varied weights and different types of failures (such as timed or randomly occurring).

Construct validity refers to the extent to which a study measures what it claims to be measuring. For RQ1 we addressed the question of whether our approach is practical for use. We applied it to two different real-world scenarios with over ten external stakeholders and described the process we followed and its ability to express a broad range of safety concerns as ISAC assumptions and pSAC constraints. Some safety concerns that we identified were considered out of scope – such as the possibility of a direct goose hit on a UAV. Before defining a safety issue as out of scope, we discussed it with our external stakeholders, in this case the local fire chief. We consider our study of developing an ISAC and pSACs as a proof-of-concept, and in future work we will design a more extensive user study of how well ISAC and pSAC creators are able to independently follow the process.

To verify the applicability of interlocking SACs, the experiments investigated the implementation of identified constraints for different application scenarios. We were able to represent all in-scope, identified assumptions as constraints with our proposed interlock approach. For RQ2 regarding the runtime monitoring evaluation we selected eight different assumptions that were observable using the SITL simulator. While both the simulated and real physical UAVs confirmed the accuracy of our interlocking SACs in a practical setting, we can not claim that the entire system is safe if no constraint is violated. We selected the parameters for the constraints based on our previous experience and discussions with key stakeholders of the urban infrastructure scenario. Our approach dynamically built a reputation model in which the UAVs lost reputation when deviations from the assumed behavior were introduced, which in turn resulted in more frequent monitoring, and the UAVs gained reputation when increased monitoring showed that constraints now were met. Again, the parameters used for the reputation model (minimum and maximum monitoring period, and forgetting factor) are based on previous experiences and experiments and need to be adapted to the specific scenario the approach is used.

8 Related Work

We discuss related work on safety assurance cases in general and UAV safety in particular, formal approaches used in the context of safety-critical systems, UAVs in urban settings, and UAV safety, as well as runtime monitoring approaches.

Safety Assurance Cases

Safety assurance cases are widely used in safety-critical domains, and requiring their use for UAV systems is under consideration by multiple governments. Our approach is consistent with the US Federal Aviation Authority’s description of what belongs in a UAV safety case [23]. The Goal Structuring Notation [11] which we use to describe ISACs and pSACs is widely used to describe SACs. It documents how a safety goal is refined and supported by evidence, as well as the context and assumptions of the goal.

Several tools exist for constructing SACs, with most based on the GSN. Tool use for developing SACs is limited with most tools being either proprietary or prototypes [68], [69], [70]. SACs need to be composable in order to scale to real cyber-physical systems. Related work has focused on identifying reusable safety case patterns or building blocks; however, instantiating the patterns and incorporating them into a specific product’s safety case continues to be largely manual [25], [71], [72], [73], [74], [75].

Contract-based reasoning provides a useful structure for composing a safety case and informs our approach. Kelly developed Modular GSN to handle the situation where one goal needs to be supported by a goal from another module, using a contract to link the two safety case modules [76]. Denney and Pai’s work proposed a formal framework to specify such contracts [77]. SACM 2.0, the OMG Structured Assurance Case Metamodel, supports such assurance-guarantee reasoning [21]. A related line of research has studied contract-based certification, where assume-guarantee obligations among the modules in a single system are mapped to the elements of the system’s safety case [78].

While previous work has been primarily directed toward producing a system-level SAC by composing safety cases for the software modules within a single system, we focus on producing an ecosystem-level SAC by composing safety cases for the independently executing systems in that ecosystem. The systems that comprise the ecosystem are fluid, with systems entering and leaving, and those systems are independently developed. Schneider et al. have
described a promising, model-based approach for communicating safety information to all companies whose products may be integrated into a particular cyber-physical system (interpreted here as entry into the urban airspace) [79].

In addition, since there are multiple systems operating concurrently in our ecosystem, the operating context for the ecosystem’s safety case has inter-dependencies with a variety of systems as well as constraints that respond to operational environment factors such as the amount of air traffic, security levels, and weather events.

Denney and Pai describe how a safety case for a UAV may need to be dynamic, i.e., to evolve when assurance variables used to monitor the operational environment, such as weather conditions or air traffic, deviate from threshold values [47]. Like us, Denney, Pai, and Whiteside [80] describe the need to broaden the safety case to consider operations. However, their work differs from ours in that we incorporate runtime monitoring evidence from operations into the ISAC.

Several approaches seek the efficiency that is potentially gained by reusing safety cases, e.g., by using domain-independent patterns and domain-specific patterns [24], [74]. Our approach similarly supports reuse since we specify interlock points at which pSACs for the individual UAVs must each provide evidence for the overarching infrastructure SAC. Wassyng et al. and Chowdhury et al., further propose to standardize safety cases using domain-specific or standardized templates [81], [82]. They argue that safety case patterns (such as [83], [84]) need to be updated less frequently than process-based standards and are easier to adopt by providing better guidance to developers. SAC templates to structure evidence have been proposed, including for use as standards [85]; however, to date they have been little studied or adopted in practice [86].

To facilitate reuse of components and of their component-level, safety case argument fragments and supporting evidence, Sljivo et al. [87] use component-level, assume-guarantee safety contracts, together with partial evidence and an explicit descriptor that more evidence should be provided, applying it to a portion of a software controller. Their approach differs from ours in that their safety contracts are at the component level, deal with component-level failure modes, and are developed independently of the context.

Oliveira et al. [88] describe tool support for generating a product line safety case architecture organized into modules that reflect feature and component boundaries. Our work shares with theirs an interest in assembling safety cases. However, they generate safety-case modules in order to potentially reuse them across a product line. In contrast, we use safety-case evidence from many independent, diverse pSACs as evidence toward assembling a much larger-scale ISAC. Calinescu et al. [89] use partial assurance arguments with placeholders for the evidence that is not available (cannot be instantiated) until runtime, applying it on a self-adaptive unmanned underwater vehicle. Our approach is similar in that we also assemble partial assurance arguments; however, while their work considers safety evidence available for a single system, our work deals with evidence regarding the safety of an infrastructure as multiple UAVs enter it, and considers different types of runtime evidence (upon entry, ongoing, upon change in reputation status).

Formal approaches

Formal approaches such as model checking for analyzing safety case content show promise but are not likely to be adopted in practice for UAVs in the near future. Significant progress toward eventual use is described in the following works. Rushby describes how automated generation of monitors to check safety properties of a system at runtime [90] can provide valuable evidence for an assurance case [91]. Kokaly et al. describe a model management framework toward reuse of assurance case components. They represent a safety case as a goal model and algorithmically assess the impact when the system specification changes [92]. Murugesan et al. examine how to formally identify sufficient trace links between requirements and the artifacts used in satisfaction arguments, toward automating the generation of such trace links [93]. Jiang, Elbaum, and Detweiler automatically infer system invariants and synthesize monitors from them, demonstrating this in the context of UAVs [94]. Groza et al. formally analyze a GSN model to detect deficits in an assurance case [95]. Hawkins et al. describe the automatic instantiation of assurance case argument patterns from information in design (AADL) and safety (FMEA) models [12]. Moosbrugger, Rozier, and Schumann employ temporal logic and Bayesian networks to detect UAV behavioral patterns indicating malicious attacks [96]. Deployment in practice of these formal approaches appears likely to face hurdles in terms of UAV developer acceptance that our approach may avoid. Several papers describe how confidence in the safety case can be measured or represented [64], [97], [98]. Unlike our approach, these do not dynamically update scores based on operational experience.

UAV in Urban Environments & Disaster Response

While much progress has been made in the image-processing and path-planning needed for the safe autonomous flight of UAVs, our work is the first that we are aware of to address safety cases for small UAVs operating in urban areas. The uses of UAVs in urban population areas are expanding rapidly, including applications for disaster management, search and rescue missions, and delivery of critical medical supplies [2], [99], [100], [101], [102], [103]. We evaluated the solutions proposed in this paper by working with local urban stakeholders to create ISACs and pSACs for two different urban UAV application areas, the first for toll-road surveillance and accident response; and the second for facilitating city-wide delivery, search and rescue in the city’s river, and hobbyist UAV activity in city parks.

UAV Safety

UAV safety is both a regulatory and technological challenge as confirmed by a recent incident review [104] concluding that “regulators need to focus primarily on airworthiness requirements”. In solving both aspects there is a clear need for a harmonized international regulatory framework [105], which is the motivation behind the International Civil
Aviation Organization’s guideline regulations for UAV [1]. These offer an appropriate stateless reference point on safe UAV operation by including subjects such as regulating airworthiness, licensing, and the airspace. However, current regulations - by ICAO and most states - do not account for fully autonomous flight such as is assumed in this work. At a technological level multiple measures are used to ensure UAV safe operation. For non-recreational outdoor activities it can be assumed that the UAV will make use of the available Global Navigation Satellite System (GNSS). As a result “Geo-fences” are a simple and effective means of keeping UAVs within air spaces that are deemed safe by the user or regulatory restrictions. For instance, UAVs made by DJI are by default programmed to not violate airspace restrictions in 17 countries regardless of user commands [106]. In addition, the presence of GNSS location awareness allows a UAV to have configurable fail-safe default behaviors such as “Return to Home” on internal low power or low and lost control signal alerts. Less common in small UAVs but an area of active research is the use of both onboard and ground-based “detect and avoid” systems. Example ground-based research systems include using RADAR [107], and Automatic Dependent Surveillance-Broadcast (ADS-B) [108] in a similar system to manned aircraft. Current research into onboard systems includes using computer vision [109], [110] and an array of active sensors such as LiDAR [111], and combinations of infrared and ultrasound sensors [112].

Monitoring & Trust

Several approaches to monitoring UAVs exist. Doherty et al. [113], for example, present a task planning and execution monitoring framework for unmanned aircraft using mission plans. The framework relies on temporal action logic to specify the behavior of the system and for reasoning about actions and changes to describe constraints (e.g., safety constraints). Machin et al. [114] propose a formal approach in the domain of autonomous systems for synthesizing monitor behavior rules. They use CTL to describe monitor properties and the SMV model-checker to validate safety strategies. However, none of these approaches combine trust and monitor, i.e., the structured collection of safety-related assumptions and constraints (e.g., in the form of a safety assurance case) and monitoring of these constraints at runtime.

More recently, in the domain of self-adaptive systems, Shevtsov et al. [115] proposed SimCA* an approach for self-adaptive systems used in the context of unmanned underwater systems. SimCA* is able to handle three different types of requirements (setpoint-, threshold- and optimized requirements) and is capable of dealing with changes of requirements at runtime. Similarly, Barbosa et al. [38] have presented Lotus®Runtime, an approach for monitoring executing traces of self-adaptive systems at runtime via Labeled Transition Systems. While our approach does not focus on self-adaptation at runtime, these approaches could complement our monitoring and reputation model with (self-)adaptive capabilities (e.g., for dynamically adapting the behavior of UAV).

Monitoring is useful within the context of safe UAV operations as it provides a mechanism by which individual UAVs can be associated with some level of trustworthiness. In complex, heterogeneous MAS (multi-agent systems) it is crucial that mischievous, malicious, or incompetent agents be avoided in order to improve overall system efficiency and safety. Thus, many studies have investigated the use of different engines to calculate the reputation of agents participating in a MAS [52], [53], [56], [116], [117], [118], [119], [120], [121].

Josang and Ismail [56] propose the beta reputation system (BRS), a trust management system based on the beta distribution function. The BRS uses the history of prior outcomes to calculate reputation and has a firm foundation in statistical theory. The BRS supports many complex reputation considerations such as combining feedback from multiple agents and “forgetting” feedback (i.e., more recent behavior is likely a better indicator of future behavior).

Griffiths [116] uses the notion of multi-dimensional trust in task delegation within cooperative MAS. He models the reputation of an agent along four dimensions: success, cost, time, and quality. In this manner, the system is able to learn specific characteristics of agents, e.g., agent A might be trusted to have high quality but cannot be trusted to finish the task on time. In situations where new agents are frequently introduced to the system it is useful to use a socio-cognitive approach that allows agents to make educated guesses about the trustworthiness of unknown agents. Burnett et al. [52] present the stereotype bootstrapping approach which provides a method for agents to generalize their experiences with previous agents in order to evaluate unknown agents.

9 CONCLUSION & FUTURE WORK

The work in this paper has proposed a new mechanism, founded on solid Software Engineering principles, for assuring safe use of UAVs in urban environments. The approach uses interlocking safety assurance cases (SACs) to target the intersection of a UAV infrastructure responsible for controlling an urban airspace and the diverse UAVs that seek to fly in it. By extending safety assurance cases with interlock points, we enable mapping of constraints imposed by the ISAC on UAVs entering or operating in its urban airspace to the compliance evidence provided by a UAV before entering the airspace or via runtime monitoring augmented by UAV reputation modeling. In addition to supporting the needs of organizations responsible for monitoring UAVs in urban environments, it directly addresses the needs of software developers building commercial, governmental, or societal applications that will fly in shared airspaces. The constraints guide them through the process of exposing a set of monitored variables, while previously developed pSACs can provide suggestions for how each of the monitorable constraints might be achieved. While rationales should be provided to explain the need for each monitored variable, the safety analysis skills required by UAV application developers are reduced.

We evaluated this approach on two urban UAV infrastructures, first by creation of ISACs and three representative pSACs for each; second by extensive simulations on these to evaluate the effectiveness of our monitoring and reputation models; and finally by testing the interlocking SAC.
implementations with multiple, varied UAVs in outdoor field studies. Exploration of two research questions addressed the applicability of the approach for different urban environments and the effectiveness of the monitoring and reputation approach for a UAV infrastructure. Experimental results showed that our implementation of interlocking safety assurance cases performed effectively and provides a promising approach for improving the identification, communication, and monitoring of safety-related constraints on UAVs flying in urban spaces.

In future work we plan to explore ways to support the evolution of ISACs using the additional information provided by runtime monitoring of the pSACs and to add self-adaptation capabilities to the infrastructure, such as allowing it to dynamically adjust weights or reputations based on current environmental conditions or the number of UAVs in the airspace. We also plan to develop tools based on the knowledge in the ISAC and sample pSACs that would support software and systems developers working in the space of UAV systems by helping them to understand the safety requirements and to explore the diverse space of design options.

ACKNOWLEDGMENTS

This project has been funded by the US National Science Foundation Grants (CCF-1741781, CCF-1649448, and CCF-1513717) and the Austrian Science Fund (FWF J3998-N31).

REFERENCES


IEEE TRANSACTIONS ON SOFTWARE ENGINEERING


Copyright (c) 2019 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.
A. Wassyng, N. K. Singh, M. Geven, N. Proscia, H. Wang, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING 19


Michael Vierhauser is a post-doctoral researcher at the Computer Science and Engineering Department at the University of Notre Dame, IN, USA. He holds a Master's degree in Software Engineering and Ph.D. in Computer Science from the Johannes Kepler University Linz. His current research interests include software product lines, safety-critical, and cyber-physical systems, and software monitoring.

Sean Bayley is currently a Ph.D student at the University of Notre Dame. He obtained his bachelor's degree and master's degree in Computer Science at California Polytechnic State University, San Luis Obispo. His research interests include software engineering and machine learning.

Jane Wyngaard is an Electrical Engineer who's research focuses on open source technologies for Earth Sciences. She holds an undergraduate degree in Mechatronics Engineering from the University of Cape Town South Africa, where she also completed her Doctorate focused on microprocessor design for Scientific High Performance Computing. Following a post-doctoral position building BigData tools for Earth Science at the NASA Jet Propulsion Laboratory, she is now working at the Center for Research Computing at the University of Notre Dame. There she is pursuing her research interests in building open source hardware and software tools for Earth and Environmental Science applications that require; Remotely Piloted Aircraft Systems and other edge computing devices for data capture, data management tools, and High Performance Computing resources.

Wandi Xiong received the BE degree in software engineering from University of Electronic Science and Technology of China in 2015. She is currently working toward the Ph.D degree in computer science at Iowa State University. Her research areas include safety-critical systems, traceability, and software product lines.

Jinghui Cheng is an Assistant Professor at Polytechnique Montreal. He obtained his Ph.D. degree at DePaul University, Chicago (2017), and was an associate research fellow at the University of Notre Dame between September 2016 and September 2017. Prior to his Ph.D. study, he accumulated years of industrial experience as a game software engineer. His research combines the fields of Human-Computer Interaction (HCI) and Software Engineering (SE). He is particularly interested in exploring technologies that support practitioners who have domain-specific expertise but special information needs, including software engineers, game designers, and medical experts.

Joshua Huseman is a graduate student completing his Master studies at the University of Notre Dame. He attained his Bachelors in Computer Science at Engineering from the University of Notre Dame in 2017. His research interests focus on the intersection of hardware and software.

Robyn Lutz is a professor of computer science at Iowa State University. She was also on the technical staff of Jet Propulsion Laboratory, California Institute of Technology, from 1983 to 2012, most recently in the Software System Engineering group. Her research interests include safety-critical software systems, product lines, and the specification and verification of molecular programmed nanosystems. She is an ACM Distinguished Scientist. She was program chair of the International Requirements Engineering Conference in 2014, general chair in 2006, and is currently on its Steering Committee. She has served two terms as an associate editor of IEEE Transactions on Software Engineering and on the editorial boards of the Journal of Software Testing, Verification and Reliability, the Journal on Software and System Modeling, and the Requirements Engineering Journal.

Jane Cleland-Huang is a Professor of Software Engineering at the University of Notre Dame. Her research interests focus upon Software and Systems Traceability for Safety-Critical Systems with a particular emphasis on the application of machine learning techniques to solve large-scale software and requirements engineering problems. She is Chair of IFIP 2.9 Working Group on Requirements Engineering, and has recently served as Associate Editor in Chief of Transactions on Software Engineering, and on the Editorial board of IEEE Software.