Author's pre-print copy accepted for publication in IEEE Computer 53 (12). Dec. 2020
The definitive version appears on IEEEXplore. DOI: 10.1109/MC.2020.3022030

(Cover Feature) Intelligent Autonomous Systems
Editors: Atif Mashkoor, Paolo Arcaini, Angelo Gargantini

Dynamic Assurance Cases: A Pathway to Trusted Autonomy

Erfan Asaadi and Dimo Petroff
KBR, Inc.

Ewen Denney, Jonathan Menzies, and Ganesh Pai
KBR, Inc. and NASA Ames Research Center

***Abstract*—We propose a system architecture that facilitates dynamic assurance of autonomous systems embedding machine learning (ML) based components, and introduce Dynamic Assurance Cases (DACs) as a generic framework to provide justified confidence in their capabilities. Underpinned by diverse evidence, rationale capture, and safety risk management principles, DACs characterize what lifecycle assurance means for such systems by capturing the technical mechanisms for managing risk, whilst elaborating the rationale for trusting autonomy.**

■ **THE PREVIOUS DECADE** has witnessed revolutionary advances in artificial intelligence (AI)-based autonomous capabilities of engineered systems, largely due to a corresponding progress in machine learning (ML) algorithms. Self-driving cars, which epitomize such AI and ML-based autonomous systems (AS) are the face of this revolution. Fatal accidents involving AS [1] have keenly highlighted the need to assure different stakeholders that systems with varying levels of autonomous capability are both *fit for purpose* and can be safely operated.

Various concurrent and ongoing standardization efforts aimed at autonomy safety analysis and assurance generally acknowledge that *assurance cases* (ACs) are a viable approach for engendering trust [2], [3]. An AC is a comprehensive, defensible, and valid justification that a system or service will function as intended for a defined application and operating environment [4].

ACs have been successfully used for safety assurance of novel aviation applications [5] where, similar to AS, regulations and safety guidelines continue to be under development. Contemporary research [6], [7] has also elaborated the merits of using ACs for addressing the varied AS assurance challenges.

AS are inherently *dynamic* and updates of their ML-based components—aimed at improving performance based on operational data—can be expected to occur reasonably often. *Online* and *lifelong* learning [8] are schemes meant specifically for this purpose, although their use has yet to be demonstrated in mainstream products. Another paradigm (the use case for this paper) employs sufficiently frequent *offline* learning to sensor data gathered during the operation of an in-service system. The re-trained ML components are then re-deployed into the operational system using, for example, *over-the-air* software updates.

The most common notion of AC is that of a *structured argument*. The objective is to convince human stakeholders, e.g., regulators, of the trustworthiness of an autonomous capability, by elaborating assurance claims that are then related by structured reasoning to a variety of supporting evidence. However, structured arguments are effectively *static*. Although arguments can in principle be evolved [9], given the role of argumentation it may be more practical for argument updates to happen offline, between missions, and when system changes are appreciable enough to warrant human decision making.

Additionally, an operational assessment of assurance can be a valuable indicator of continued fitness for purpose, potentially facilitating intervention and recovery when there is insufficient assurance. Such a *dynamic assessment and update* of assurance that is aimed at machine consumption must necessarily be in a computable form, e.g., as a quantification, or by using a formal language, such as a logic.

Moreover, once a baseline level of acceptable risk has been established for a system approved for operation, from an assurance standpoint the focus is on maintaining that baseline. An architectural viewpoint of the assurance mechanisms implemented in the system, along with computable forms of assurance, are inherently better suited for this purpose, being more tightly coupled to the physical and functional system architecture.

In our perspective, a richer AC concept including both static and dynamic elements is required instead: namely, a rigorously defined Dynamic Assurance Case (DAC). Our concept is distinctly different from prior work [9], [10], comprising the following core interrelated aspects: *assurance rationale* (captured using structured arguments as in traditional, static ACs) which is informed by *assurance policies* and an *assurance architecture*, and which is underpinned by *evidence*, together with *assurance measures* facilitating confidence quantification that serve as an architectural mechanism for dynamic assurance. Collectively, these offer a multi-viewpoint, model-based approach for through-life assurance of an AS. Each core element models a different aspect of assurance that we will explain subsequently. As more is discovered about the AS and its environment, we update the DAC components as appropriate (some in real-time, and others over a longer interval) keeping the models consistent via consistency relations (not described here) that also enable inter-model traceability.

For the subsequent discussion, the ensuing terminology is relevant: *assurance* is the provision of justified confidence that an *item*—i.e., a component, system, or service—possess the required assurance properties. An *assurance property* is a logical, possibly probabilistic characteristic associated with *assurance concerns* (or *assurance objectives*), i.e., functional capabilities and dependability attributes. An *assurance claim* results from applying one or more assurance properties to a particular item. Practically, an assurance claim can be considered to be equivalent to a requirement that has been (or will eventually be) substantiated by concrete evidence.

DYNAMIC ASSURANCE

Architecture

The concept of *assurance measure* characterizes the extent of confidence in an assurance property through a probabilistic quantification of uncertainty. As we will later see, it represents a computable notion of confidence tied to the wider assurance concerns that are substantiated by the (system-focused) DAC. As such, it encodes a baseline level of acceptable risk for a deployed AS, based on a suitably abstract, probabilistic model of the same.

Our concept of *dynamic assurance* integrates assurance measures into an AS, to facilitate run-time confidence assessment of its assurance properties. That, in turn, can support subsequent decision-making for continued assurance of AS fitness for purpose in operation.

Figure 1 shows our proposed architecture for trustworthy autonomy: a collection of *run-time monitors* assess system properties (which may include assurance properties) taking inputs from the environment and system state. The assurance measure quantifies the confidence/uncertainty in the assurance properties using both the inputs and outputs of the monitors. This, in turn, is one of the inputs for a decision mechanism that determines whether to proceed with the nominal system

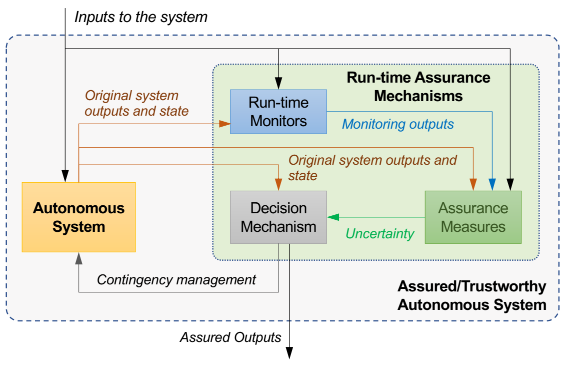

Figure 1. An architecture for trustworthy AS.

operation or to invoke contingency management actions. For example, when there is insufficient confidence, any expected system output that is otherwise *assured* may be masked. We concretize these concepts later by application to an aviation domain AS.

The aim of *run-time assurance*, also known as *run-time verification*, is to provide updates as to whether a system satisfies specified properties as it executes [11]. Typically, this uses run-time monitors, which evaluate the properties using values extracted from the system and environment state. In a sense, therefore, the notion of assurance measure we have described here is a kind of monitor. However, it is worth making the following distinctions:

- monitors typically relate directly to properties of the system, whereas an assurance measure characterizes *confidence* in our knowledge of such properties; and
- an assurance measure seeks to aggregatea range of sources of information, including monitors. Thus it can be seen as a form of *data fusion*.

The architecture in Figure 1 is also closely related to the *simplex architecture* and its variants [11], though there are some differences: since the assurance measure models the AS, which may itself be implemented as a simplex architecture, assurance measure outputs can be viewed as providing an additional level of analytic redundancy that is wider in scope that the safety controllers that simplex traditionally employs. We believe this can be advantageous in a run-time tradeoff between performance and safety. Also, in simplex, decision making takes environment state as one of the inputs, whereas here they are reflected in the uncertainty forecast from assurance measures.

Methodology

We now clarify the relationship of assurance measures to the DAC concept and its core components (described in detail in the next section). **Figure 2** shows a high-level methodology of the lifecycle of developing a DAC for an AS (broadly considered as the physical and logical system descriptions and its concept of operations).

First we establish a baseline level for sufficient assurance, largely, by developing a static AC focused on the *system requiring assurance*. As shown (in Figure 2 in the box labelled ‘System Focus’), this comprises various core components, i.e., assurance policies, architecture, rationale, evidence, and quantification models (especially relevant at this stage).

Besides quantifying a pre-deployment assurance baseline, together with the hazard analysis it enables us to identify and discriminate between properties for which we construct assurance measures, and those that require monitoring. Trivially, these include properties whose violation is expected to impede continued safe operation or prevent mission completion. However, for instance, monitoring may suffice for certain component-level properties verified in design under assumptions of system and environment states, while assurance measures may be better suited to system-level properties potentially affected by emergent behavior. In general, we assume that there exist run-time monitors, some of which are part of the system requiring assurance, while others are tied to the validity of the evidence items used.

We compile the corresponding quantification models into optimized executables—assurance measures—that we then integrate into the system architecture as described earlier. Via a dashboard, the assurance measure can also passively provide a real-time assessment of the confidence in assurance properties to end users. In either case, since this modifies the system design there is a need to provide additional justified confidence in both the efficacy of the assurance measure itself, and in the integration.

For this, we develop *additional* static assurance artifacts (see Figure 2, the box labeled

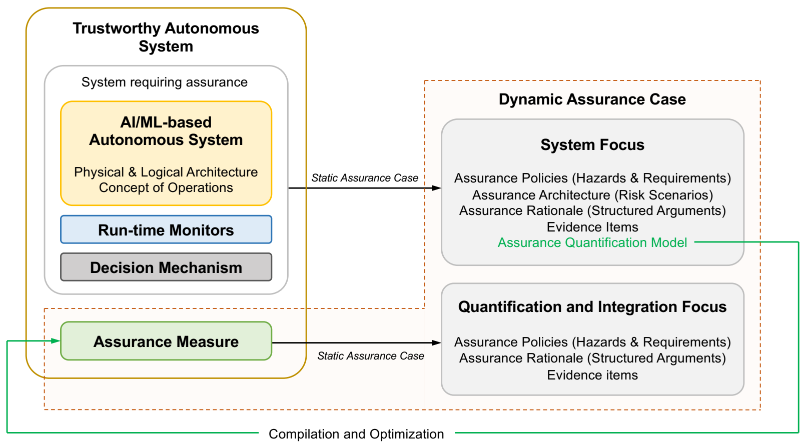
Trustworthy Autonomous System
System requiring assurance
AI/ML-based Autonomous System
Physical & Logical Architecture
Concept of Operations
Run-time Monitors
Decision Mechanism
Assurance Measure
Static Assurance Case
Static Assurance Case
Dynamic Assurance Case
System Focus
Assurance Policies (Hazards & Requirements)
Assurance Architecture (Risk Scenarios)
Assurance Rationale (Structured Arguments)
Evidence Items
Assurance Quantification Model
Quantification and Integration Focus
Assurance Policies (Hazards & Requirements)
Assurance Rationale (Structured Arguments)
Evidence items
Compilation and Optimization

Figure 2. Dynamic assurance methodology.

'Quantification and Integration Focus') for objectives such as timeliness of the assurance measure in the context of recovery actions; assurance measure performance in terms of forecast sensitivity and specificity; and mitigation of hazardous interactions due to assurance measure integration.

DYNAMIC ASSURANCE CASES

Our concept of DAC is the combination of the static assurance artifacts (focused on the system, quantification and integration), and the assurance measure, which provides dynamic assurance (Figure 2).

Practically, through-life assurance has a broad scope, and a comprehensive DAC must address a plurality of *core* and *supplementary* assurance concerns [5] through one or more of its main, interrelated components (see Figure 2). We consider these components to be part of an *assurance toolkit*, where the particular assurance concern being addressed informs which components are required, and impacts their size and complexity.

For instance, AS system safety is a core concern—itself covering a broad gamut of assurance objectives including but not limited to design safety, and operational safety—that requires all DAC components. In contrast, reliable compilation of an ML model into a platform-specific executable has a narrower, tool-qualification focus. It represents a *supplementary assurance concern* that requires fewer DAC components.

We now describe each DAC component, their role in (dynamic) assurance, and their interrelations.

Assurance Policy Model (APM)

An *assurance policy* concretely expresses what AS assurance means in terms of: *i*) the conditions under which assurance is impacted, in particular where there is a higher risk of undesired effects, and *ii*) the requirements for mitigating the risk associated with those impacts. As previously mentioned, these requirements are, in fact, assurance claims that are yet to be substantiated by evidence. As such, they capture both assurance properties, and their allocations to the relevant AS items.

The APM is a model-based representation of assurance policies providing a basis against which sufficiency of assurance can be established. As such, it is both related to and kept consistent with other core DAC components, as we will clarify when describing the latter. More generally, the APM captures the (functional and non-functional) guarantees to be provided together with the assumptions made, mappings to the AS physical and logical components, bounds on acceptable behaviors, system states, etc.

To formulate assurance policies, we can leverage traditional hazard analysis techniques, such as a Functional Hazard Analysis (FHA), or newer ones such as System Theoretic Process Analysis (STPA), along with requirements decomposition and refinement techniques.

Assurance Architecture Model (AAM)

An *assurance architecture* models a system from an assurance viewpoint as a collection of scenarios that show how risk is modified.

The AAM is a model-based representation

of the assurance architecture. We have adopted *barrier models*, extending an earlier notion of *safety architecture* [12], to represent the AAM. This choice has been motivated by the observation that a collection of scenarios can conveniently describe an assurance concern and, in turn, the related assurance properties. This tightly couples the AAM to the physical and functional items constituting the AS architecture, whilst highlighting the roles that the items and their capabilities play in risk modification, e.g., prevention, recovery, tolerance, masking, etc.

Conceptually, the AAM composes distinct but related operational scenarios, each of which models the impact on assurance in terms of: *i*) the progression of events that migrate the system to higher risk states; and *ii*) the mechanisms of the system architecture employed to manage risk (barriers). We use Bow Tie Diagrams (BTDs) to specify these scenarios in a graphical way [12].

Shared BTD elements capture relations between scenarios at appropriate abstraction levels. Thus, for system-level assurance concerns, we model system-level operational risk situations, each of which we can further refine into lower-level risk scenarios that themselves require design-time or operational mitigations.

The AAM can be seen as an *implementation* of the APM, with each model being closely related to, and synchronized with the other, whilst recording different information. For example, we can model the assurance architecture of the barriers themselves, as additional BTDs scenarios showing barrier failure modes and their local effects. Simultaneously, the APM captures the corresponding assurance requirements that ought not to be captured in the AAM.

Thus, a value addition of the AAM is the assurance viewpoint into the system architecture, through which the contribution of the latter to sufficient assurance can be explicitly highlighted.

Assurance Quantification Model (AQM)

The main purpose of assurance quantification is to assess the confidence that can be justifiably placed in an (AS) item based on data associated with measurable assurance properties. Consequently, an appropriate AQM can help both to establish a baseline level of assurance (to support the decision to release a system into service), and to evaluate whether or not that level continues to be maintained in operation (supporting a run-time risk assessment and mitigation).

Many AS used in safety-critical applications are stochastic dynamical systems. As such, the system-level AQM is a suitably abstract, probabilistic, model-based representation of a stochastic process whose underlying random variables (RVs) describe the AS state space. Specific realizations of those RVs correspond to the assurance properties of interest, and the associated probability distributions reflect the uncertainty in those properties. That is, we express *confidence* in terms of the uncertainty in assurance properties, with lower uncertainty corresponding to higher confidence that the related assurance property holds. Effectively, this is a probabilistic query on the AQM, leveraging a range of techniques for uncertainty quantification (UQ) and propagation that account for various types of uncertainty, e.g., model and parameter uncertainty.

Component focused assurance quantification of ML components is also feasible and the corresponding AQM takes into account component-level usage details [13]. We can relate a component level AQM to the system-level AQM, though we will not consider it further here.

Evidence Model (EM)

Evidence underpins assurance, and is crucial for trusting AS. The EM as a core DAC component relates heterogenous evidence items, records their provenance, captures the assertions that can be made, and their usage context, whilst facilitating their tracing to other core DAC components. In particular, we link evidence items to assurance rationale (described next) for both justifying specific AS assurance claims, and to substantiate why the evidence items should themselves be trusted. In the latter case, note that the assurance claims are about the evidence items, whereas in the former case, they are about the AS.

More generally, the EM is an interface to reference concrete external evidence items in the application-specific DAC components. For instance, when a structured argument references, say, a piece of formal verification, it refers to the corresponding entry in the EM.

Assurance Rationale

Structured arguments capture various kinds of assurance rationale, in the same way as a traditional, static AC. This includes not only the reasoning why specific (AS) assurance claims should be accepted based on the evidence supplied, but also rationale substantiating supplementary assurance concerns of the remaining DAC components themselves; e.g., sufficiency of the stated assurance policies; appropriateness of assumptions such as independence of risk mitigations in the assurance architecture; the relevance and completeness of the scenarios specified in the AAM; and, as mentioned earlier, the suitability and relevance of the evidence used.

We can communicate such rationale as a narrative, in a tabular or a graphical form, or as a combination of the three. Here, we use the Goal Structuring Notation (GSN) [14]: a standardized graphical language to describe key components of an assurance argument. For methodological details on developing assurance arguments, see [4].

EXAMPLE

We now concretize the preceding concepts by application to an aviation domain AS supplied by our industrial collaborators: an Unmanned Aircraft System (UAS) embedding a generic autonomous taxiing capability, intended for ground movement operations at airports. We will present illustrative excerpts and fragments that are not intended to be exhaustive.

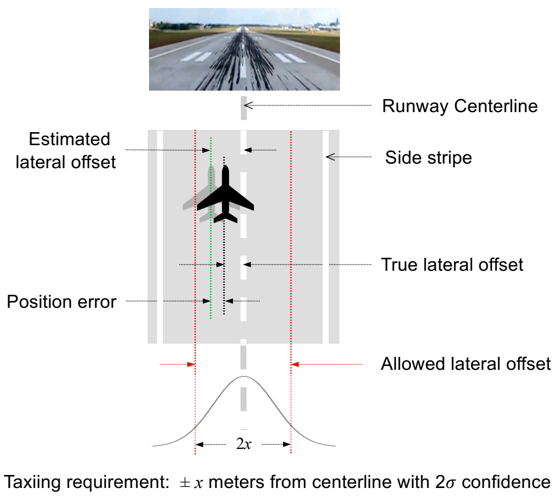

Figure 3. Autonomous runway centerline tracking.

Autonomous Taxiing System Description

We are primarily interested in providing assurance of *runway centerline tracking*. **Figure 3** shows this use case, where the UAS is required to taxi along the runway, following the runway centerline without both *i*) violating an allowed lateral offset (assured performance), and *ii*) lateral runway overruns, i.e., veering off the sides of the runway (assured safety). A simplified system architecture to realize this autonomous taxiing capability comprises, firstly, a deep Convolutional Neural Network (CNN) that implements a *perception function*, ingesting video images from a wing-mounted camera pointed to the nose of the aircraft. This ML component performs regression under supervised learning producing as output, estimates of *cross track error* (CTE), i.e., the horizontal distance between the aircraft nose wheel and the runway centerline. Next, a classical Proportional-Integral-Derivative (PID) controller then uses CTE estimates to generate the appropriate steering and actuation signals.

Autonomous Taxiing Assurance Policies

Safety and performance are amongst the two main assurance concerns relevant for autonomous taxiing. The APM records the hazards to these (and other) concerns, i.e., the operational activities and system conditions that, under certain environmental conditions, compromise safety and performance. It additionally specifies assurance requirements that embed the relevant assurance properties. Those, in turn, give the system states under which the hazards are implausible (ideally, untrue). Thus, a candidate performance assurance property is (informally) to *maintain the allowed lateral offset from the centerline whilst taxiing* (see Figure 3). Similarly, a candidate safety assurance property is to *not depart the runway pavement boundaries by crossing the side stripes whilst taxiing*.

The APM also contains potential precursors (causes) and effects, along with candidate mitigations and the associated requirements. For instance, some candidate causes of the hazards to performance and safety include *hazard contribution modes* (HCMs) of the ML components [15], e.g., *accurate CTE estimates from runway markings other than the centerline*. Mitigations can include, for example, a combination of *i*) *system-level* *prevention and recovery devices*, e.g., emergency brakes, or redundant aircraft localization; and *ii*) *design modifications* aimed at improving ML component accuracy, e.g., retraining the ML component using training data augmented with test data from prior or lower-accuracy variants.

Autonomous Taxiing Assurance Architecture

Figure 4 shows a BTD illustrating the risk posed to safety and performance in performing a *hazardous activity* using ML-based autonomy: taxiing at a speed of 25 knots on a wet runway under low visibility conditions at dusk, when there are no crosswinds.

Here, a violation of the performance assurance property (the central, *top* event) can lead to an unsafe consequence, signifying a compromise of the safety assurance property. This scenario shows two potential initiating events (*threats*, in BTD terminology), one of which is a functional deviation (labelled Threat 1), while the other is a hazardous control action (labelled Threat 2).

Interspersed between the events are mitigations (*barriers*, in BTD terminology) that modify risk in specific ways: here, a combination of redundancy, failover mechanisms for the perception and control components, runtime monitoring, and emergency braking, invoked to assure that taxiing is both safe and performant. The extent of risk modification that a barrier provides is reflected quantitatively as *barrier integrity* [12].

Figure 4 can be seen as a snapshot of a partially developed AAM, which we can further refine based on the assessment of residual risk levels (RRLs) of the relevant top events and consequences. For instance, if the RRL is at an unacceptable level, additional risk mitigations such as safety margins may be used.[1]

We can develop this AAM and refine the associated scenario-driven analysis not only at system level (as discussed here), but also at a lower, component-level, thereby relating (system-level) operational assurance concerns to (component-level) design assurance objectives.

Assurance Arguments and Evidence

Figure 5 shows a fragment of a structured argument in GSN capturing the assurance rationale for autonomous taxiing safety. The text labels identify the GSN language elements, links with solid arrowheads represent inferential relations while those with hollow arrowheads indicate contextual relations. The diamond decoration on nodes indicate that the reasoning is to be further developed (see [14] for more details).

The argument in Figure 5 combines disparate forms of argumentation: a qualitative, inductive argument (the fragments including strategies S2, S5 and downwards) reasoning over hazard identification, and mitigation (e.g., through prevention barriers), and the rationale underlying quantification (as highlighted in Figure 5).

The qualitative argument captures the (static) reasoning that is implicit in the assurance architecture. However, it includes additional substantiation on fitness for purpose of the barriers (e.g., the claims in the as yet undeveloped sub-goal nodes G6, G7 and G8). The quantitative argument fragment relates quantified assurance claims as determined from the AQM (i.e., as shown in sub-goal nodes G10–G12) to the wider argument, serving as a complimentary branch to the qualitative argument leg.

The fragment here is focused on the system level, but connects to claims on components. For example, the argument justifying the claim in sub-goal node G9 (not shown here) would address ML component contribution to the unsafe event in the root goal node (G1), invoking the mitigation of the applicable hazard contribution modes (HCMs) [15], recorded in the APM.

The evidence used (not shown) includes both formal and descriptive environment and aircraft system models, scenario descriptions, simulation results, formal safety specifications and verification results (e.g., of reachability properties).

Dynamic Assurance of Autonomous Taxiing

The assurance measure implements the AQM, providing a real-time quantitative forecast of the uncertainty that the aircraft violates its performance and safety assurance properties whilst taxiing.

The AQM here is a Dynamic Bayesian Network (DBN) that specifies the joint distribution over a collection of RVs (including those for the true and estimated CTE), whose states correspond to the temporal evolution of the states of the

[1] In Figure 4, node and label colors give an intuitive visualization of risk, e.g., red/orange colors reflect higher risk.

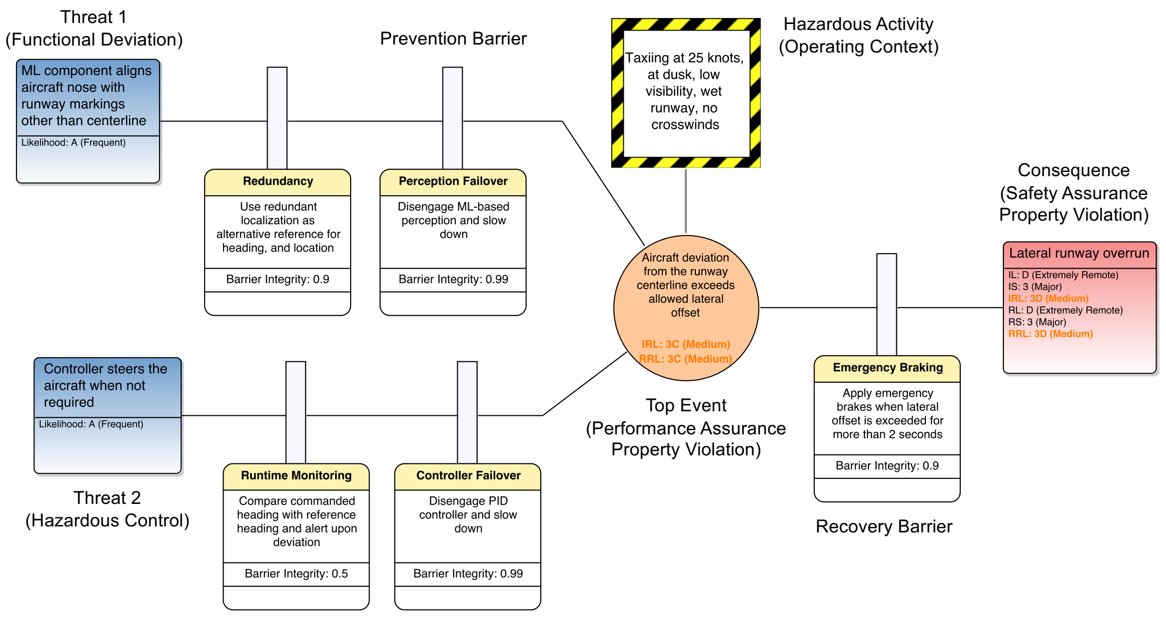
Threat 1
(Functional Deviation)
ML component aligns aircraft nose with runway markings other than centerline
Likelihood: A (Frequent)
Prevention Barrier
Redundancy
Use redundant localization as alternative reference for heading, and location
Barrier Integrity: 0.9
Perception Failover
Disengage ML-based perception and slow down
Barrier Integrity: 0.99
Taxiing at 25 knots, at dusk, low visibility, wet runway, no crosswinds
Hazardous Activity
(Operating Context)
Aircraft deviation from the runway centerline exceeds allowed lateral offset
IRL: 3C (Medium)
RRL: 3C (Medium)
Top Event
(Performance Assurance Property Violation)
Consequence
(Safety Assurance Property Violation)
Lateral runway overrun
IL: D (Extremely Remote)
IS: 3 (Major)
IRL: 3D (Medium)
RL: D (Extremely Remote)
RS: 3 (Major)
RRL: 3D (Medium)
Emergency Braking
Apply emergency brakes when lateral offset is exceeded for more than 2 seconds
Barrier Integrity: 0.9
Recovery Barrier
Controller steers the aircraft when not required
Likelihood: A (Frequent)
Threat 2
(Hazardous Control)
Runtime Monitoring
Compare commanded heading with reference heading and alert upon deviation
Barrier Integrity: 0.5
Controller Failover
Disengage PID controller and slow down
Barrier Integrity: 0.99

Figure 4. A view of the AAM: BTD showing a performance and safety violation scenario.

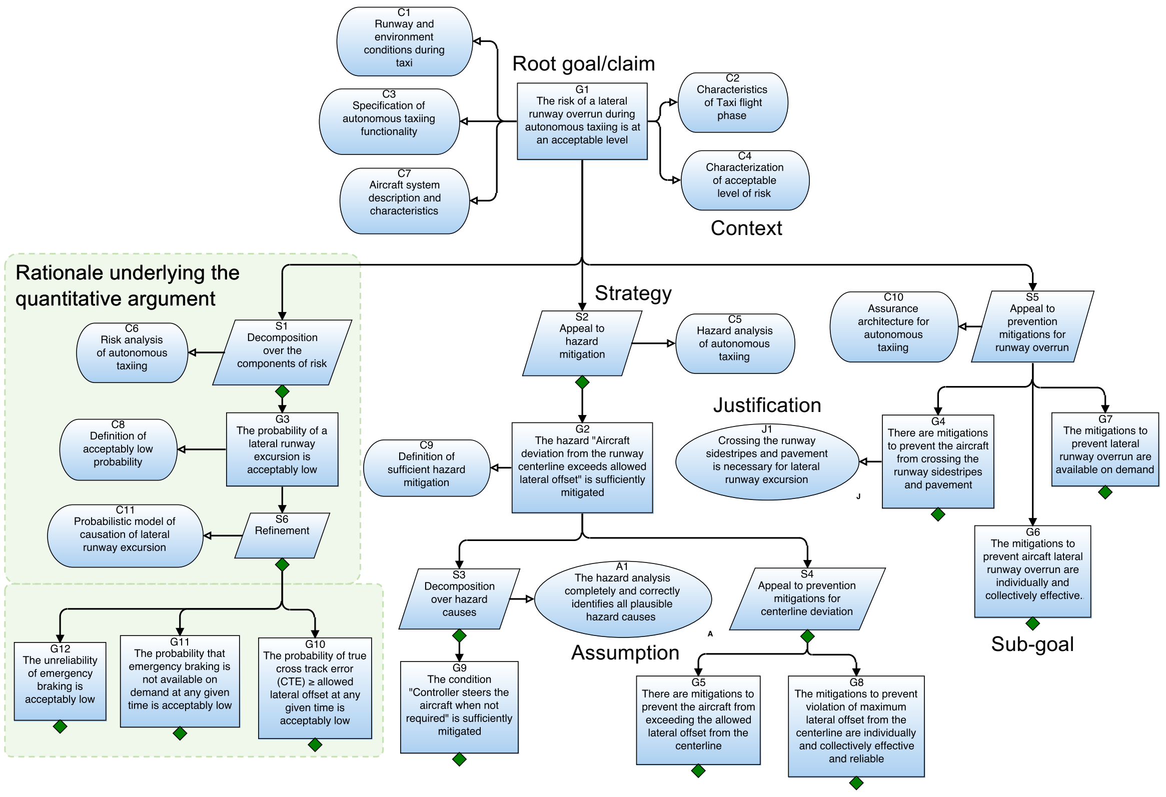
C1
Runway and environment conditions during taxi
Root goal/claim
G1
The risk of a lateral runway overrun during autonomous taxiing is at an acceptable level
C2
Characteristics of Taxi flight phase
C3
Specification of autonomous taxiing functionality
C4
Characterization of acceptable level of risk
C7
Aircraft system description and characteristics
Context
Rationale underlying the quantitative argument
S1
Decomposition over the components of risk
C6
Risk analysis of autonomous taxiing
Strategy
S2
Appeal to hazard mitigation
C5
Hazard analysis of autonomous taxiing
C10
Assurance architecture for autonomous taxiing
S5
Appeal to prevention mitigations for runway overrun
C8
Definition of acceptably low probability
G3
The probability of a lateral runway excursion is acceptably low
C9
Definition of sufficient hazard mitigation
G2
The hazard "Aircraft deviation from the runway centerline exceeds allowed lateral offset" is sufficiently mitigated
Justification
J1
Crossing the runway sidestripes and pavement is necessary for lateral runway excursion
J
G4
There are mitigations to prevent the aircraft from crossing the runway sidestripes and pavement
G7
The mitigations to prevent lateral runway overrun are available on demand
C11
Probabilistic model of causation of lateral runway excursion
S6
Refinement
G6
The mitigations to prevent aircraft lateral runway overrun are individually and collectively effective.
S3
Decomposition over hazard causes
A1
The hazard analysis completely and correctly identifies all plausible hazard causes
A
S4
Appeal to prevention mitigations for centerline deviation
Sub-goal
G12
The unreliability of emergency braking is acceptably low
G11
The probability that emergency braking is not available on demand at any given time is acceptably low
G10
The probability of true cross track error (CTE) ≥ allowed lateral offset at any given time is acceptably low
Assumption
G9
The condition "Controller steers the aircraft when not required" is sufficiently mitigated
G5
There are mitigations to prevent the aircraft from exceeding the allowed lateral offset from the centerline
G8
The mitigations to prevent violation of maximum lateral offset from the centerline are individually and collectively effective and reliable

Figure 5. Excerpt of a system-level structured argument in GSN.

taxiing aircraft. Quantifying assured performance amounts to querying the DBN for the posterior probability that the true aircraft location (i.e., true CTE) is within the bounds of the allowed lateral offset.

Based on thresholds that balance safety and performance (as determined by our end-user industrial collaborators), the decision mechanism (see Figure 1) uses the assurance forecast to decide whether the aircraft should continue, slow down, or engage the emergency brakes and stop. The assurance measure includes the inputs and outputs of a run-time monitor that determines if the camera output images are from outside the training distribution of the the ML controller.

We have integrated both the assurance measure and the run-time monitor into a hardware-in-the-loop simulator, visualizing various properties of interest and the corresponding assurance quantification on a user dashboard (not shown here). For more details on the implemented run-time monitor, assurance measure, and its visualization, see [13] and [16].

To provide confidence in the efficacy of the assurance measure, we augment specific components of the autonomous taxiing DAC, as described earlier (see the Methodology section). Specifically, a supplementary assurance argument leverages validation evidence [16] to make the case that the assurance measure is a reasonably accurate abstraction of the time-series behavior of autonomous taxiing, and that together with the run-time monitor produces uncertainty estimates that are conservative and consistent when operating in an uncertain environment.

CONCLUSIONS

Dynamic Assurance Cases (DACs) offer a pathway towards through-life assurance for AI and ML-based autonomous systems. They are intended to provide both dynamic assurance at run-time using assurance measures, and design-time assurance aimed at certification. The former provides operational situational awareness to humans as well as online machine processing, while the latter comprises more static artifacts that can be evolved to maintain certification over the longer-term. There are a range of possible realizations of the DAC concept, and we have described one such instantiation, and its rigorous foundations. The main novelty is providing assurance from multiple viewpoints, including one that supports dynamic assurance update in a form suitable for machine consumption.

One key focus for future work is in quantification, and we are exploring how best to integrate assurance measures into system-level decision making, whilst providing domain-specific visualization of assurance measures for end users.

To facilitate adoption of the DAC concept into practice we are also developing supporting infrastructure, i.e., languages, models, reusable assurance artifacts, and automation technologies in our assurance case tool, AdvoCATE [4]. Our overarching goal is to build a generic, integrated assurance framework to provide justified confidence first during design, and then continually in operation. Our concept is compatible with emerging standards for safety assurance of AS, whilst enhancing their basic assurance mechanism.

ACKNOWLEDGMENT

This work was supported by the Defense Advanced Research Projects Agency (DARPA) under the Assured Autonomy Program. The opinions, findings, recommendations or conclusions expressed are those of the authors and do not represent the official views or policies of DARPA, the Department of Defense, and the United States Government.

■ REFERENCES

1. National Transportation Safety Board, “Collision between vehicle controlled by developmental automated driving system and pedestrian Tempe, Arizona March 18, 2018,” NTSB, Washington, DC, Highway Accident Report NTSB/HAR-19/03, November 2019.
2. International Organization for Standardization, “Road vehicles — Safety of the intended functionality,” Standard ISO/PAS 21448:2019, January 2019.
3. Underwriter Laboratories Inc., “Standard for Safety for the Evaluation of Autonomous Products UL 4600,” April 2020.
4. E. Denney and G. Pai, “Tool Support for Assurance Case Development,” *J. Autom. Soft. Eng.*, vol. 25, no. 3, pp. 435–499, September 2018.
5. R. Clothier, E. Denney, and G. Pai, “Making a Risk Informed Safety Case for Small Unmanned Aircraft System Operations,” in *Proc. 17th AIAA Aviation Technology, Integration, and Operations Conference*, June 2017.

6. J. E. McDermid, Y. Jia, and I. Habli, “Towards a Framework for Safety Assurance of Autonomous Systems,” in *Proc. AAAI Workshop on AI Safety*, CEUR Workshop Proceedings, January 2019.
7. R. Bloomfield, H. Khlaaf, P. Ryan Conmy, and G. Fletcher, “Disruptive innovations and disruptive assurance: Assuring machine learning and autonomy,” *IEEE Computer*, vol. 52, no. 9, pp. 82–89, September 2019.
8. D. L. Silver, Q. Yang, and L. Li, “Lifelong machine learning systems: Beyond learning algorithms,” in *2013 AAAI Spring Symposium Series*, 2013.
9. E. Denney, I. Habli, and G. Pai, “Dynamic Safety Cases for Through-Life Safety Assurance,” in *2015 IEEE/ACM 37th Intl. Conf. Soft. Eng.*, vol. 2, May 2015, pp. 587–590.
10. R. Calinescu, D. Weyns, S. Gerasimou, M. U. Iftikhar, I. Habli, and T. Kelly, “Engineering Trustworthy Self-Adaptive Software with Dynamic Assurance Cases,” *IEEE Trans. Soft. Eng.*, vol. 44, no. 11, pp. 1039–1069, November 2018.
11. Subcommittee F38.01 on Airworthiness, *ASTM-F3269-17, Standard Practice for Methods to Safely Bound Flight Behavior of Unmanned Aircraft Systems Containing Complex Functions*, ASTM International, 2017.
12. E. Denney, G. Pai, and I. Whiteside, “The Role of Safety Architectures in Aviation Safety Cases,” *Rel. Eng. Sys. Safety*, vol. 191, 2019.
13. E. Asaadi, E. Denney, and G. Pai, “Towards Quantification of Assurance for Learning-Enabled Components,” in *Proc. 2019 15th European Dependable Computing Conf.* September 2019, pp. 55–62.
14. The Assurance Case Working Group, “Goal Structuring Notation Community Standard Version 2,” Jan. 2018.
15. E. Denney, G. Pai, and C. Smith, “Hazard Contribution Modes of Machine Learning Components,” in *Proc. AAAI Workshop on AI Safety*, CEUR Workshop Proceedings, January 2020.
16. E. Asaadi, E. Denney, and G. Pai, “Quantifying Assurance in Learning-enabled Systems,” in *Proc. 39th Intl. Conf. Comp. Safety, Reliability, and Security*. September 2020.

Erfan Asaadi is with KBR, Inc. His research interests are in the topics of uncertainty quantification, probabilistic machine learning, and deep learning. Erfan holds a Ph.D. in Mechanical Engineering from the University of Pretoria. Contact him at erfan.asaadi@us.kbr.com.

Ewen Denney is with KBR, Inc., and NASA Ames Research Center. His research interests cover safety and mission assurance, including automated tool support and formal foundations. Ewen holds a Ph.D. in Computer Science from the University of Edinburgh and he is a member of the IEEE. Contact him at ewen.denney@nasa.gov.

Jonathan Menzies is with KBR, Inc., and NASA Ames Research Center. He is interested in the application of software engineering techniques to assurance case tool development. Jonathan holds a B.Sc. in Computer Science from Heriot-Watt University. Contact him at jonathan.menzies@nasa.gov.

Ganesh Pai is with KBR, Inc., and NASA Ames Research Center. His research interests lie in the broad areas of safety, dependability, and mission assurance, as applied to aerospace systems and software. Ganesh holds a Ph.D. in Computer Engineering from the University of Virginia, and he is a Senior Member of the IEEE. Contact him at ganesh.pai@nasa.gov.

Dimo Petroff is with KBR, Inc. Dimo is interested in domain specific languages, model-based development, user-interface engineering, and their application to assurance case tool development. He holds an M.Sc. in Electronics and Computer Science from the University of Edinburgh. Contact him at dimo.petroff@us.kbr.com.