Weaving the Web of Assurance: AOPG as a Foundation for Coherent Safety Arguments

Peter Karpati[a], Xueli Gao[a], and Norbert Carte[b]
[a] Institute for Energy Technology, Halden, Norway, Peter.Karpati@ife.no, Xueli.Gao@ife.no
[b] United States Nuclear Regulatory Commission, Rockville, MD, USA, Norbert.Carte@nrc.gov; *The views expressed herein are the author's and do not represent an official position of the U.S. NRC.*

Abstract: Aspect-Oriented Property Graphs (AOPG) paradigm represents safety assurance information as a typed property graph that links risk contributors, properties, allocated obligations, realization mechanisms, verification activities, and evidence in a single authoritative substrate. This paper presents advanced AOPG features that address common maintenance problems in structured safety cases: duplication from cross-cutting concerns, drift as designs evolve, and manual rework of reviewer-facing argument views. We introduce deterministic aspect weaving with policy-driven conflict resolution, query-based governance for coverage/closure/freshness and change-impact checks, and rule-based generation of Goal Structuring Notation (GSN)-style arguments from explicit mapping rules. Together, these mechanisms support repeatable, model-centric, human-governed assurance while preserving familiar reviewer-friendly views as on-demand projections of the underlying graph. For the PSAM community, AOPG complements PRA-style frequency-and-consequence reasoning by making systematic causes of assurance weakness (such as deterministic deficiencies in specification, design, integration, and V&V) and their traceable paths to safety claims explicit in a governed model.

1. INTRODUCTION

Safety assurance for digital systems can rely on structured argumentation to demonstrate that safety requirements are satisfied by system design and supported by credible evidence. Structured Safety Assurance Cases (SSACs) [1,2] are often expressed using tree-based notations such as Goal Structuring Notation (GSN) [3] or related formalisms. SSACs have become an accepted medium for communication and their conceptual foundation – explicitly linking safety claims, reasoning, and evidence – is sound and widely recognized.

However, experience from safety demonstration practice shows that, while the idea of structured argumentation is powerful, its practical realization faces some limitations. These limitations become particularly pronounced for complex digital instrumentation and control (DI&C) systems, where safety assurance must address not only functional correctness but also architectural constraints, timing behavior, independence, common-cause failure, cybersecurity, and other cross-cutting concerns. As systems and assurance scope grow, traditional document- and tree-centered approaches increasingly struggle to scale without introducing duplication, inconsistency, and maintenance overhead.

This perspective is also relevant to the PSAM community. PRA commonly focuses on failure modes, consequences, and likelihood, whereas many assurance questions for digital safety systems concern **systematic causes**: deterministic deficiencies in specification, design, integration, verification and validation, or related engineering decisions that can defeat a safety claim. These issues are causal and structural rather than primarily statistical. AOPG does not replace PRA, rather, it complements it by making such causes, their mitigations, and their logical paths to safety properties explicit and queryable. This distinction is also recognized in U.S. nuclear practice: alongside random failures, 10 CFR 50 Appendix A explicitly notes the need to consider "systematic, nonrandom, concurrent failures" of

redundant protection-system elements, highlighting that adequate safety also depends on identifying and removing underlying systematic causes.

A first set of challenges arises from **how assurance cases (ACs) are constructed and maintained**. Traditional SSAC approaches are difficult to learn, in part because they introduce terminology and diagrammatic constraints that differ from everyday engineering practice. As a result, assurance is often treated as a parallel documentation activity rather than being co-developed with the design. Even though standards and notations exist, they provide limited domain-specific guidance, leading to ad-hoc argument structures that depend heavily on individual experience. Most tool support focuses on diagram editing rather than on systematically constructing, governing, or checking assurance content.

A second set of challenges concerns **integration with engineering artefacts**. ACs are often assembled after the fact, drawing selectively from design descriptions, analyses, and tests. Claims, architectural elements, and evidence therefore remain only loosely connected, making it difficult to detect when design changes invalidate parts of the AC. Over time, arguments risk drifting away from the engineering reality they are meant to represent.

Beyond these procedural issues, **tree-based representations introduce fundamental structural limitations** [4]. Safety assurance reasoning is inherently interrelated: requirements converge on shared components, mitigations interact, and constraints cut across architectural boundaries. Tree structures force this information into hierarchical branches, emphasizing and deemphasizing selected relationships, and leads to duplicated representation across branches or remaining implicit. As a result, ACs can give a misleading impression of completeness while hiding critical dependencies. These issues are amplified in performance-based assurance focusing on desired, measurable outcomes, where reasoning must connect goals, architecture, behavior, and evidence in a coherent and maintainable way.

These limitations motivated a shift in how safety assurance information is represented. The **Aspect-Oriented Property Graph (AOPG) paradigm** [5] is based on the observation that if assurance knowledge is inherently interrelated and tightly coupled to engineering artefacts, then its primary representation should also be explicitly interconnected. In AOPG, safety risks, assurance properties, design allocations, realization mechanisms, verification activities, and evidence are captured as typed and explicitly related elements in a single directed property graph, which serves as the authoritative source of assurance information. Familiar artefacts such as GSN arguments are derived from this graph rather than maintained as primary documents. This paradigm shift addresses the representational mismatch between tree-based arguments and the underlying interrelated structure of assurance reasoning, while improving integration with design-time artefacts.

However, a directed property graph-based representation alone is not sufficient. Large assurance graphs can become complex, and without rigorous semantics and disciplined mechanisms they risk accumulating duplication or losing coherence as systems evolve. This motivated the **advanced AOPG features** described in this paper. Recurrent cross-cutting obligations led to the introduction of **aspect weaving**, allowing such concerns to be applied systematically without manual duplication. The need for consistent oversight and efficient review motivated **query-based governance**, enabling automated checks for coverage, completeness, evidence adequacy, and change impact. Finally, the continued importance of established assurance artefacts motivated **structured view generation**, in which familiar GSN arguments are assembled from the graph using explicit mapping rules and patterns. Together, these features aim to elevate safety assurance from a document-maintenance activity to a model-based, human-governed process that remains aligned with engineering reality, scales to cross-cutting concerns, and supports risk-informed review without compromising clarity.

The remainder of this paper is organized as follows. Section 2 introduces typed property graphs and summarizes the AOPG approach and related work. Section 3 presents aspect weaving and conflict resolution. Section 4 describes query-based governance. Section 5 explains deterministic GSN view generation. Section 6 discusses adoption and GraphRAG-style infrastructures, and Section 7 concludes.

2. BACKGROUND AND RELATED WORK

This section provides the conceptual background required for the advanced AOPG features presented later. We first introduce typed property graphs, the AOPG approach, its ontology, and maturity. It is followed by a review of related efforts through the lens of the advanced mechanisms addressed.

2.1 Typed Property Graphs and the AOPG Approach

AOPG is founded on the use of a **typed property graph** as the authoritative representation of safety assurance information. A property graph is a directed graph structure in which nodes and edges may carry attributes ("properties"). In a typed property graph, every node and edge additionally belongs to an explicitly defined type that constrains its permitted attributes, relations, and multiplicities, enabling structural consistency and enforceable semantics.

The **AOPG approach** instantiates this paradigm for safety assurance. It defines a conceptual ontology consisting of node types (see Table 1) and typed relations (see Figure 1). Together, these elements capture the trace from risk contributors, through required system properties and allocated obligations, down to concrete realization mechanisms and verification evidence.

Table 1: AOPG ontology: node types

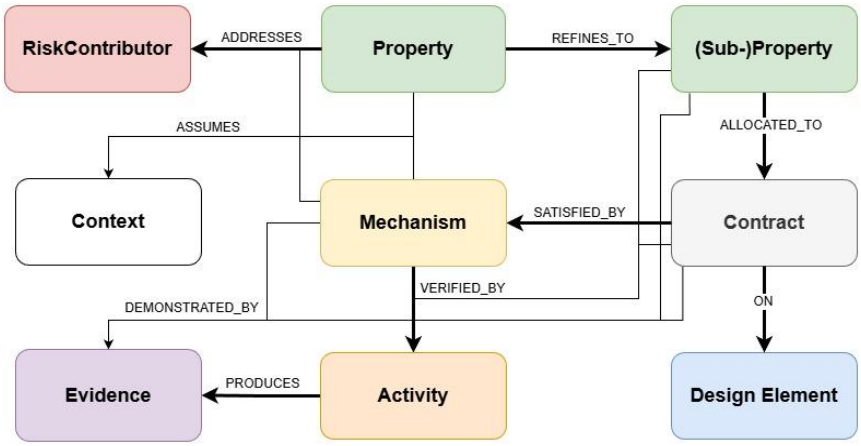
Node type	Short description	Example attribute types
RiskContributor	A node representing a specific contributor to system risk, such as a hazard, hazardous condition, event leading to a hazard, cause, or accident.	kind, severity (for hazard, accident), likelihood (for cause, event, condition), source, status
Context	Conditions or assumptions under which the argument holds (e.g., operating environment).	mode, environment, operational design domain (ODD), constraints
Property	An attribute, quality, or characteristic of something e.g., a SAC may include a claim that a particular property must hold for the system to be considered safe (e.g., "Single-failure tolerance").	type, criticality, status, originAspect, (optional) constraint describing attribute types
Contract	Obligations or guarantees at a system or component level. This is like requirements and the context under which they must be met.	assumptions, guarantees, integrityLevel, verificationLevel
DesignElement	The actual piece of the system that implements the contract.	type, domain, integrityLevel, vendor/version
Mechanism	The technical means or feature that achieves the property (e.g., redundancy, watchdog timer).	type, parameters, originAspect
Activity	Actions taken to ensure or verify compliance (e.g., testing, analysis).	method, standard, responsible
Evidence	Concrete artifacts containing the information supporting a property/contract/mechanism (e.g., test reports, analysis results).	artifactType, location, date, validityWindow, provenance
Justification	Rationale explaining why the evidence and properties/contracts/mechanisms are adequate.	text, references

AOPG deliberately separates **assurance representation** from **assurance presentation**. The graph is the authoritative source of truth; familiar artefacts such as GSN arguments are generated as projections rather than maintained manually. This allows the same assurance base to support multiple stakeholder views while remaining structurally coherent and up-to-date. For example, reviewers may prefer concise GSN views, engineers may inspect design allocations and realization mechanisms, and governance users may rely on queries or dashboards over the same substrate.

The approach also includes a **pattern-based workflow** that guides the incremental construction of the assurance graph, from seeding risk contributors and contexts, through property refinement and allocation, to mechanism realization and evidence capture. These patterns encode good practice and

make structural completeness checkable. The workflow is designed to **co-develop the AOPG substrate with the engineering artifacts** such as system architecture models, safety analyses, V&V artefacts.

Figure 1: AOPG relations and workflow (both partial, bold arrows representing the flow)



AOPG originated from an **AI-assisted rapid prototyping** [6] effort. Microsoft Copilot was used as a co-creative partner to explore ontology designs, refinement patterns, weaving semantics, and governance queries in parallel, while avoiding conceptual drift. This process enabled rapid validation of ideas using illustrative cases without committing prematurely to a toolchain. As a result, the current state of **AOPG is conceptual and experimental**. Its ontology, patterns, and operational semantics are being exercised on representative examples and subject of collecting expert feedback, but industrial-scale validation and tooling remain future work. AOPG is therefore not a finished product at this stage, but a consolidated and explicit foundation for structured, model-based assurance.

2.2 Related Work and Motivation for Advanced Features

The advanced features discussed in this paper draw on, and extend, several strands of related work.

Structured Assurance and Metamodels: Tree-based notations such as GSN [3] and CAE [7] dominate assurance practice and provide well-established visual languages for presenting arguments. Their limitations with respect to cross-cutting concerns, scalability, and maintenance are widely recognized. The Structured Assurance Case Metamodel (SACM) [2] addresses interoperability and exchange of assurance information but remains largely presentation-oriented and does not prescribe a workflow or governance mechanisms. AOPG is compatible with these notations at the view level but deliberately moves the authoritative model to a graph substrate.

Automation, Queries, and Governance: Several approaches emphasise automation and analysis over assurance structures. Tools such as AdvoCATE [8], ARCOS/CertGATE [9], and related environments use patterns, queries, or knowledge bases to assess coverage, sufficiency, and evidence adequacy. These efforts strongly motivate AOPG's query-based governance but typically externalize the assurance substrate into separate ontologies or repositories. AOPG integrates governance queries directly over the same graph used to construct and project the assurance argument.

The Assurance 2.0 [10] and CLARISSA [11] lines of work further strengthen reasoning rigor by introducing defeaters, confidence, and soundness considerations. AOPG is complementary to these approaches: rather than redefining argument semantics, it focuses on making assurance information structurally explicit, queryable, and amenable to governance checks, thereby providing a stable substrate on which richer reasoning layers may later be applied.

Design-Anchored Assurance and Weaving: Closer integration between assurance and design is pursued by approaches such as Resolute [12, 13], which evaluates assurance rules directly over architecture

models, and model-weaving approaches [14, 15], which bind assurance cases to design and process models through explicit correspondences. These efforts strongly influenced AOPG's focus on co-development with engineering artefacts and on preventing drift between assurance and design.

AOPG differs in where weaving occurs. Instead of weaving assurance artefacts into design models or generating arguments from lifecycle models, AOPG performs aspect weaving at the assurance-graph level, injecting cross-cutting obligations into the assurance model itself based on binding queries, with explicit precedence and provenance.

Similarly motivated work: Workflow+ [16, 17] targets many of the same assurance-maintenance challenges as AOPG – scalability, repeatability, and reducing ad-hoc GSN structures – by treating assurance as a model-management and conformance problem. Workflow+ models process/control flow, data/work products, and constraint/argument derivation. It frames assurance in terms of conformance/refinement between normative guidance, a defined safety engineering process, and its execution for a specific system. AOPG shares many goals of Workflow+ but places the primary representation in a typed knowledge graph and uses query-based governance plus aspect weaving for cross-cutting concerns; GSN arguments are generated as views from this substrate

In addition, there is a mature, industrial suite of digital applications called ABB Ability™ SafetyInsight™ [18]. Both AOPG and SafetyInsight aim to improve confidence that safety risks are understood, addressed and supported by evidence. SafetyInsight™ is a digital process safety management suite for high-hazard process industries that digitalizes the safety lifecycle by creating a process-safety digital twin from Engineering Technology data and contextualizing it with IT/OT operational data. It includes lifecycle modules such as HAZOP/LOPA and Safety Instrumented Function design, and supports IEC 61508/61511-aligned governance and reporting across engineering and operations. AOPG, by contrast, proposes a model-centric assurance paradigm in which assurance knowledge itself is explicitly represented, governed, and evolved, enabling its deterministic handling.

3. ASPECT WEAVING AND DETERMINISTIC CONFLICT RESOLUTION

Aspect weaving operates directly on the authoritative AOPG substrate and injects additional typed nodes and relations where the concern is applicable. Each injected element carries provenance (e.g., originating aspect and weave-run identifier), enabling auditability, deterministic re-weaving after change, and selective impact analysis when an aspect policy evolves. Although it may introduce repeated assurance elements across the graph, this differs from problematic duplication in tree-based assurance cases because it results from systematic rule application, not manual copy-and-paste. The repetition reflects legitimate multiplicity of obligations while maintenance effort remains low through reweaving. Duplications in generated GSN views are therefore a presentation artefact, not modelling debt. Next, the aspect-weaving mechanism is detailed (see also Figure 2).

3.1 Binding and Weaving

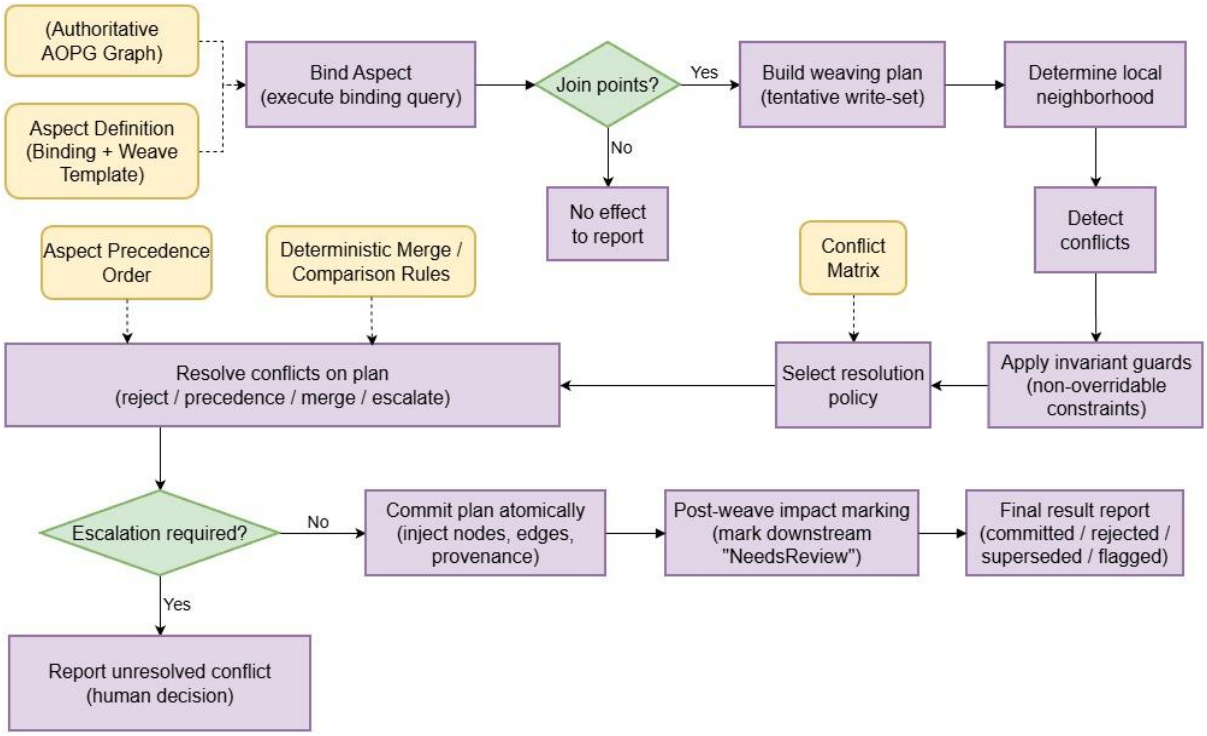
An aspect in AOPG encapsulates a reusable assurance policy for a specific cross-cutting concern. Conceptually, it consists of two parts: **binding logic** identifies where in the assurance graph the concern applies while **weaving logic** specifies and applies modifications on the AOPG substrate. Aspect weaving is incremental: when the graph evolves, its re-execution updates only the affected portions.

The applicability of an aspect is determined by a **binding query** (e.g., "All safety-classified digital DesignElements", "All redundant channel groups") that is intentionally structural: it is evaluated over the assurance graph itself and over explicitly represented attributes, rather than over external artifacts such as code or documents. This makes the applicability of an aspect explicit, reviewable, and repeatable, but requires that assurance-relevant characteristics of the engineering artefacts be represented directly in the model. The binding query returns a set of **join points**, typically DesignElements, Contracts, or Properties, indicating where the aspect's assurance logic applies. In

advanced cases, a join point may be a typed relation when the semantics of that relation are themselves the intended target of the aspect, but for the baseline mechanism node-level join points are sufficient.

For each join point, weaving proceeds by constructing a tentative **weaving plan** that describes the graph modifications required by the aspect. This plan consists of a set of proposed operations, including node additions, relation additions, and explicitly declared attribute assignments. The plan is not applied immediately; instead, it is checked for consistency against existing assurance content to ensure that no obligations are silently overwritten or weakened, by using a conflict resolution mechanism.

Figure 2: Aspect-weaving workflow including the conflict resolution mechanism



3.2 Conflict Resolution

Conflict detection is performed locally, within a bounded neighborhood of the join point. The neighborhood comprises the join point itself, its directly connected assurance elements within one hop over assurance-relevant relations, and the semantics of those relations. This locality keeps weaving scalable while still capturing the region where incompatible obligations can arise. Conflicts are identified when proposed injections contradict existing constraints, overwrite attribute values, or attempt to modify semantic properties of relations in an incompatible way.

A central strength of the mechanism is its explicit treatment of **constraints**. In AOPG, a constraint is an assurance-level restriction that bounds allowable behavior, structure, assumptions, or evidence expectations at a given part of the assurance graph, thereby limiting what subsequent weaving or refinement may assert without explicit resolution. They are not modeled as a separate ontology class; instead, they are represented as Properties that play a constraint role. **Constraint-carrying Properties** are explicitly marked and described using typed attributes that make deterministic comparison possible, e.g., by specifying the constraint *category*, *form*, *operator*, and *value*. This ensures that decisions such as "which constraint is stricter" can be made algorithmically and are not left to interpretation.

Once potential **conflicts** are identified, **resolution** follows a deterministic policy-driven process. First, **hard invariants** are enforced: any existing constraint marked as non-overridable cannot be weakened by aspect weaving, and conflicting injections are immediately rejected. For remaining conflicts, a

conflict matrix is consulted. The conflict matrix is an external, declarative policy artifact that maps combinations of aspects (and, where relevant, constraint categories) to resolution policies such as *rejection*, *precedence-based resolution*, or *deterministic merging*. **Aspect precedence** itself is declared as a global, acyclic, and total order over all aspects considered for the system; this order is validated before weaving and must be stable during execution. Aspect precedence is not universal or tool-defined; it is a system-specific assurance decision that must be configured, reviewed, and justified as part of the assurance setup. While a single global order is often sufficient, heterogeneous systems may require scope-dependent precedence, where conflicts are resolved at concrete graph locations using local state.

When **precedence-based resolution** applies, the authoritative contribution is determined by the declared order. Lower-precedence injections are suppressed, while higher-precedence contributions may override existing, overridable content. Such overrides are only logical: overridden elements are not removed but are marked as superseded, preserving provenance and auditability. Operational queries and views filter on effective content so scalability and comprehensibility are maintained. Where **merging** is permitted, it is performed only according to explicitly defined deterministic rules, and if no such rule exists, the situation is escalated for human decision.

If all conflicts are either absent or resolved deterministically, the **weaving plan is committed** *atomically* to the assurance graph. This results in a new, enriched assurance state that integrates the aspect's contributions without ambiguity. In cases where weaving introduces or strengthens constraints, a subsequent **change-impact step** marks downstream assurance elements as requiring review. This does not invalidate them automatically, rather, it ensures that *governance queries* surface affected elements so that human experts can assess adequacy under the new constraints.

3.3 Example: Cybersecurity vs AI Diagnostics Aspects

To illustrate the mechanism, consider a system comprising a **safety-domain protection function** and a strictly **one-way telemetry path** implemented via a gateway and data diode, exporting operational data to an external diagnostics function. A *cybersecurity aspect* is applied first. It binds to the telemetry chokepoint and injects assurance content stating that safety-relevant data exported from the protection system (e.g., status or measurement values used for monitoring) is protected against modification or spoofing, together with a constraint-role Property that explicitly enforces one-way communication and is marked as non-overridable. Subsequently, an *AI diagnostics aspect* is applied. While this aspect legitimately applies to the external diagnostics component, it also attempts to introduce interactive acknowledgement semantics on the same telemetry path. During aspect weaving, the local conflict check at the telemetry join point detects that the proposed interaction contradicts the existing one-way communication constraint. Because this constraint is designated as non-overridable, the *conflicting portion* of the AI-related injection is deterministically rejected, while the remainder of the AI assurance obligations can still be woven into the diagnostics part of the system. The resulting assurance graph thus preserves the intended architectural isolation while allowing both cybersecurity and AI-related concerns to coexist in a controlled and explicit manner.

Overall, the **aspect-weaving mechanism in AOPG** achieves a balance between automation and governance. It enables systematic reuse of cross-cutting assurance logic while ensuring that conflicts are handled explicitly, precedence is respected, invariants are protected, and downstream impacts are surfaced rather than hidden. This makes aspect weaving a practical and trustworthy technique for evolving complex assurance cases under multiple interacting concerns.

4. QUERY-BASED GOVERNANCE AND REVIEW SUPPORT

By **governance**, we mean the disciplined, repeatable control of the assurance model so it remains internally consistent (no structural contradictions), complete against its declared obligations (all required links exist), and traceably supported by acceptable evidence (obligations connect to current, policy-valid evidence) as the system and assurance evolve. In AOPG, it appears as systematic, repeatable oversight of assurance completeness, consistency, and evidential support through explicit

queries over the AOPG substrate. Rather than relying on implicit reviewer judgement or manual inspection of documents, governance expectations are made concrete in the model and checked directly.

Governance queries evaluate whether structural and evidential obligations encoded in the model are met. This shifts governance from a document-checking activity to a model-based oversight process: queries objectively expose gaps and inconsistencies, while decisions about acceptability and sufficiency remain with human reviewers. To ensure uniform scrutiny, the same governance queries apply regardless of whether assurance elements were manually authored or injected via aspect weaving.

A first class of governance queries addresses **coverage and closure**: whether required assurance relationships are present and structurally complete. Because the AOPG ontology encodes expected relations explicitly, such checks can be expressed directly as graph traversals. Typical **coverage queries** identify, RiskContributors not addressed by any Property, Properties that have not been allocated to Contracts, or Contracts without realizing Mechanism. **Closure queries** focus on the lower parts of the assurance chain, on such assurance elements that are not linked to verification Activities or Evidence. Such queries can be executed repeatedly at defined lifecycle checkpoints and scoped to subsystems or concerns. *They do not claim that assurance is adequate*; rather, they surface structural omissions that might otherwise remain hidden in large, evolving assurance models.

A second class of governance queries focuses on **evidence adequacy and temporal validity**. Evidence nodes carry explicit metadata, allowing queries to identify expired tests, outdated analyses, or obligations supported only by stale evidence. By making validity expectations explicit and queryable, AOPG provides an objective basis for performance-based review without relying on undocumented assumptions. It is particularly important in long-lived systems and iterative development contexts, where design changes or operational updates can silently invalidate previously acceptable evidence.

Change-impact queries operate by tracing explicitly modelled dependency paths from a well-defined change anchor – such as a modified Context, DesignElement, Contract, or constraint-carrying Property – through typed assurance relations in the AOPG graph. Rather than performing global or heuristic analysis, these queries rely on *bounded, semantics-preserving traversals* to identify which Properties, verification Activities, Evidence, and argument fragments are downstream of the change. The role of change-impact analysis is therefore not to assess adequacy automatically, but to deterministically surface which assurance elements require reconsideration. This disciplined, traversal-based mechanism is particularly important for aspect weaving: when cross-cutting obligations are injected or constraints are strengthened, change-impact queries ensure that their consequences remain visible and reviewable, preventing automated augmentation from masking the need for explicit human reassessment.

Together, these governance queries establish whether the AOPG substrate is **structurally complete and evidentially current** that is a natural prerequisite for generating coherent, reviewer-facing argument views, which is addressed next.

5. GSN-VIEW GENERATION AND ARGUMENT ASSEMBLY

AOPG supports on-demand **depiction of safety arguments** as GSN diagrams. These diagrams are generated views: the AOPG graph remains the authoritative source of assurance information, while GSN is a projection assembled from explicit graph elements and relations. This **separation of representation and presentation** avoids manual maintenance of tree arguments and reduces drift: when the graph evolves through refinement, aspect weaving, or change handling, the regenerated GSN view reflects the current governed state. View generation does not introduce new assurance semantics: every GSN element shown in the view corresponds to an explicitly modeled AOPG node or relation.

5.1 Argument Assembly

Argument assembly applies explicit **mapping rules** that translate AOPG structures into GSN elements and link types. The process begins by selecting an **anchor** and then **traversing** assurance-relevant relations in a fixed order to build a structured argument. The steps are as follows (see also Figure 1):

1. Anchor and root goal phrasing. The anchor is either (i) a top-level Property or (ii) a RiskContributor. If a RiskContributor is used, the root GSN Goal is phrased using the attributes of the ADDRESSES relation: the RiskContributor is partially or completely controlled/prevented/mitigated.
2. Claim decomposition from Properties. Property nodes map to GSN Goals. Logical decomposition is driven by REFINES_TO relations between Properties, which define the goal–subgoal hierarchy. The resulting tree constitutes the "upper" argument skeleton, independent of architectural binding.
3. Obligation and realization chain. For leaf Properties, the generator traverses ALLOCATED_TO to create obligation-level goals for Contracts, then ON to bind each Contract to its responsible DesignElement. Realization is then made explicit by traversing SATISFIED_BY: each realizing Mechanism becomes a subordinate Goal under the corresponding Contract goal. In this way, the generated view demonstrates how the system is intended to satisfy safety objectives.
4. Verification and evidence closure. AOPG represents verification actions explicitly. Property/Contract/Mechanism nodes link to verification Activities via VERIFIED_BY, and Activities produce Evidence artifacts via PRODUCES. In the GSN view, verification is represented as an explicit Goal layer: an "Activity performed and adequate" goal supports the corresponding claim and is itself supported by Solution nodes referencing the Evidence produced by that Activity. This design keeps provenance explicit and avoids treating evidence without an associated verification action. Optional Justification nodes (JUSTIFIES → Property / Contract / Mechanism) can be projected verbatim as GSN Justification elements attached to the corresponding claim, preserving rationale without requiring non-deterministic interpretation.
5. Context integration. Operating bounds and assumptions are projected by mapping ASSUMES relations to GSN Context nodes attached to the relevant Goals (and where applicable, to reasoning Strategies). This ensures that mode, environment, and other conditions remain visible to reviewers and that claims are not interpreted outside their intended scope.

Because assembly is rule-based and repeatable, multiple GSN views can be generated from the same AOPG substrate without modifying the underlying model.

5.2 Role of GSN Patterns

GSN Strategy nodes are not stored explicitly in AOPG. Instead, they are generated from recognized AOPG structures using a small, version-controlled strategy registry. At minimum, the generator may insert a Strategy between consecutive claim levels (goal-subgoal) and between claims and verification/evidence layers to make the reasoning step explicit. Strategy labels are selected deterministically based on the structural pattern being instantiated, such as refinement-based decomposition (from REFINES_TO), allocation-based reasoning (from ALLOCATED_TO/ON), realization-based reasoning (from SATISFIED_BY), and verification-based reasoning (from VERIFIED_BY/PRODUCES/DEMONSTRATED_BY). This keeps the ontology compact (presentation constructs are not embedded in the assurance substrate) while still producing readable and reviewer-familiar argument steps in the generated view.

A limitation is that meaningful Strategy text cannot always be inferred from structure alone: in the absence of additional typing or pattern metadata, generated Strategy labels may remain generic (e.g., "Argue by refinement" or "Argue by verification"). This is a deliberate trade-off in favor of determinism and traceability: the generator does not invent semantics, and richer strategy explanations can be supplied through the external registry or by adding explicit pattern tags to the graph where needed.

6. DISCUSSION

AOPG is positioned as a **model-centric assurance substrate**, not a monolithic tool. Its ontology and mechanisms are exercised on representative examples, but broader empirical validation and tool-specific representation are still needed. The approach's contribution lies in making assurance

knowledge explicit in a typed property graph, enabling deterministic aspect weaving, query-based governance, and on-demand generation of views from a single source of truth. Commercial tools exist for representing typed property graphs in a database and using graph-based query languages to extract information from it while enabling the representation of other design information in the same database.

6.1 Practical considerations

AOPG provides a generic, application-independent ontology. Practical use requires instantiating it for a concrete system and, where needed, specializing attributes, constraints, and governance queries to reflect domain-specific assurance expectations. While conceptually tool-agnostic, effective adoption benefits from graph persistence, validators, governance dashboards, and deterministic view generation. The approach assumes a shift from document-centric assurance to **model-first assurance**, where GSN diagrams are generated views rather than maintained artifacts. AOPG does not replace hazard analysis, or PSA; these can populate RiskContributor and Context elements, while AOPG preserves traceability to design obligations, realization mechanisms, and evidence within a single governed model.

Different stakeholders may need to interrogate different parts of the same assurance knowledge represented by the graph. For example, developers and system designers may focus on allocations and realization mechanisms; safety engineers on the trace from risk contributors to properties, activities, and evidence; security experts on cross-cutting aspect insertions and conflicts; operational users on evidence validity, change impact paths, or later lifecycle status; and reviewers on generated GSN views anchored in different claims or hazards. AOPG supports these **differing needs** through query and projection mechanisms over a governed graph, avoiding separate assurance artefacts that drift apart.

A limitation of generated GSN views is that users who interact only with the projection may miss relationships, cross-links, or context that remain explicit in the underlying graph, or may overinterpret the tree as the full reasoning structure. The generated GSN should therefore be treated as a reviewer-facing presentation, while the AOPG substrate remains the authoritative source to consult when additional structure, provenance, or evidence status must be examined.

These points map directly to the three concerns raised in the introduction. AOPG reduces **construction and maintenance** burden through reusable patterns, aspect weaving, and deterministic regeneration. Further, it improves **integration** by linking assurance elements directly to design allocations, mechanisms, activities, and evidence. Finally, it addresses **structural limitations** by representing many-to-many assurance dependencies explicitly in the graph rather than forcing them into a tree. From a **PSAM perspective**, this means that AOPG can complement probabilistic models by preserving the explicit causal and engineering rationale behind safety claims, especially where the critical concern is not only failure likelihood but also the elimination or control of systematic defeaters.

6.2 AOPG and GraphRAG-based AI infrastructures

Recent GraphRAG approaches combine **knowledge graphs with retrieval-augmented generation** (RAG) to move beyond vector-only RAG, enabling multi-hop reasoning, explainability, and structured grounding for LLM-based and agentic systems. Neo4j's GraphRAG framework [19] can rely on a graph schema to guide extraction and querying.

The AOPG ontology can be reused as a domain-specific schema for safety-assurance GraphRAG pipelines. Compared to typical lightweight extraction schemes (labels and relations only), AOPG offers richer semantics (i.e., explicit roles, attributes, and constraints) supporting consistency, provenance, and governance in safety-critical contexts. Two application patterns are plausible:

1. *Deterministic AOPG mode*: Assurance information is authored and queried deterministically as part of ongoing development, LLMs are excluded from the critical path.
2. *GraphRAG-assisted mode*: LLMs support extraction, querying, and explanation over an AOPG-constrained graph, with explicit validation and human review gates.

These modes align with different lifecycle phases: AOPG's native workflow supports co-development with engineering, while GraphRAG is particularly useful after the fact, when existing documentation must be structured, explored, or explained. In both cases, **acceptability remains a human judgement**; automation serves to surface structure, gaps, and relationships rather than to decide adequacy.

6.3 Next steps and future directions

As an initial, regulator-oriented step, we plan to evaluate AOPG as an **analysis aid for existing public licensing applications**. The AOPG ontology will serve as an extraction scheme for an LLM (with human entry where needed) to populate a Neo4j [20] graph database with safety assurance information from the documents. The subsequent review is performed via **deterministic graph queries** (e.g., Cypher), so that findings such as missing nodes, unsupported claims, incomplete evidence chains, or gaps in reasoning can be reproduced and audited. The core evaluation question is whether this graph-based workflow helps reviewers identify issues **faster or more reliably** than reading the documents alone, including by systematically challenging arguments (reducing confirmation bias) rather than only confirming them. In parallel, we plan comparative experiments by constructing alternative graph schemas (based on e.g., CAE, GSN, SACM) and assessing how each representation constrains the queries that can be asked and the issues that can be surfaced.

A limitation of this experiment setup is that the **LLM-based extraction/population step is itself a variable**, so the populated graph may diverge from what the documents actually state, and extraction fidelity may differ across schemas. Also, results are **not directly comparable across extraction schemes** because each ontology constrains what can be represented and therefore what deterministic queries can be asked. We address this by treating extraction quality and schema expressiveness as explicit factors (lightweight spot-checks and "same reviewer question intents" mapped per schema). This will help to iteratively refine the ontology and the query set based on what reviewers cannot yet express or surface, before moving on to later lifecycle (co-development) use.

In the future, AOPG could be extended into two adjacent application modes by adding a small set of **operational and lifecycle properties**. First, as a basis for a **safety management system** across the plant lifecycle, the AOPG model could be augmented with properties that capture long term safety-relevant updates, e.g., updated reliability/availability assumptions, maintenance burden indicators, and "still-valid"/"needs-update" status. This way, the governance queries (coverage, closure, validity) can be rerun when numbers change and when evidence or assumptions must be refreshed. Second, as a basis for a digital-twin-like **operational safety status assessment**. The same graph can be supplemented with operational state properties (e.g., operable/inoperable, degraded, out of service/maintenance) and then queried to assess the safety status of the current configuration. Both extensions reuse the same explicit risk-to-obligation-to-equipment links and simply add state/lifecycle properties, something tree-based notations like CAE and GSN cannot support as a queryable, property-bearing model.

7. CONCLUSIONS

This paper presented advanced features of **Aspect-Oriented Property Graphs (AOPG)** that address recurring assurance maintenance problems in large, evolving safety cases: duplication of cross-cutting obligations, drift between assurance views and engineering reality, and manual rework of reviewer-facing argument trees. AOPG treats the typed assurance graph as the authoritative substrate and generates familiar artefacts such as GSN as deterministic projections rather than maintained documents, thereby reducing synchronization burden and improving repeatability.

We described three complementary mechanisms. First, **aspect weaving** enables systematic, provenance-preserving injection of cross-cutting assurance content at structurally identified join points, while **deterministic conflict resolution** (precedence and policy-driven handling of incompatible injections) prevents silent weakening of obligations. Second, **query-based governance** operationalizes oversight through explicit checks for coverage, closure, evidence validity/freshness, and change impact, ensuring that gaps and stale support are surfaced consistently as the system evolves. Third, **structured**

GSN-view generation assembles reviewer-friendly arguments by traversing typed relations in a fixed order and applying explicit mapping rules, optionally projecting recorded justifications where available.

AOPG is currently conceptual and experimental, broader empirical validation and tool-specific representation remain future work. Nevertheless, the approach provides a consolidated foundation for model-centric, human-governed assurance that remains aligned with engineering artefacts and scales without sacrificing reviewer-familiar communication views.

Acknowledgment

The authors thank the members of the **Halden HTO Project** for their support, especially **US NRC** experts for their constructive input. This work was funded and supported by the OECD Nuclear Energy Agency (NEA) Halden Human-Technology-Organization Project (https://ife.no/en/project/oecd-nea-halden-hto-project/).

References

[1] P. Karpati, X. Gao, and N. C. Carte, "*Connecting safety standards to safety*", OECD NEA Halden HTO Project Report, Report no. HTOR-005, 2023.
[2] Object Management Group (OMG), "*Structured Assurance Case Metamodel (SACM) Version 2.3*", Formal/23-05-08, 2023. https://www.omg.org/spec/SACM/2.3/PDF, accessed: 15.01.2025.
[3] SCSC Assurance Case Working Group, "*Goal Structuring Notation Community Standard Version 3*", 2021, https://scsc.uk/r141C:1?t=1, accessed: 19.12.2025.
[4] P. Karpati, X. Gao, and N. C. Carte, "*Structured safety argumentation for a nuclear reactor protection system – an assessor's view*", OECD NEA Halden HTO Project, Report no. HTOR-027, 2025.
[5] P. Karpati, X. Gao, N. C. Carte., "*Aspect-Oriented Property Graphs: A Paradigm Shift in Safety Assurance*", In proceedings of R.A.M.S. Europe 2026, to be published.
[6] P. Karpati, X. Gao, N. Carte., "*From Zero to Graph: How AI Built a Safety Assurance Paradigm*", In proceedings of R.A.M.S. Europe 2026, to be published.
[7] Adelard (NCC Group), "*Claims, Arguments, Evidence (CAE) framework*", https://claimsargumentsevidence.org/notations/claims-arguments-evidence-cae/, accessed: 11.01.2026.
[8] E. Denney, G. Pai, and J. Pohl, "*AdvoCATE: An Assurance Case Automation Toolset*", FAA REDAC report, 2015.
[9] ARCOS, "*CertGATE*", https://arcos-tools.org/tools/certgate, accessed: 15.01.2026.
[10] J. Rushby and R. Bloomfield, "*Assurance 2.0: A Manifesto*", Proceedings of the 29th Safety-Critical Systems Symposium, 2021.
[11] S. Varadarajan et al., "*CLARISSA: Foundations, Tools & Automation for Assurance Cases*", 42nd AIAA/IEEE Digital Avionics Systems Conference (DASC), Oct 2023.
[12] A. Gacek et al., "*Resolute: An Assurance Case Language for Architecture Models*", HILT 2014; preprint arXiv:1409.4629
[13] Loonwerks, "*Resolute: an assurance case language for architecture models*", http://loonwerks.com/tools/resolute.html, accessed Jan 2026.
[14] R. Hawkins et al., "*Weaving an Assurance Case from Design: A Model Based Approach*", HASE 2015.
[15] R. Hawkins, I. Habli, T. Kelly, "*The Need for a Weaving Model in Assurance Case Automation*", White Rose Technical Note, 2017.
[16] Z. Diskin, N. Annable, A. Wassyng, and M. Lawford, "*Assurance via Workflow+ Modelling and Conformance*", 2019, DOI: 10.48550.
[17] N. Annable et al., "*Generating Assurance Cases Using Workflow+ Models*", SAFECOMP 2022, LNCS 13414, 2022.
[18] ABB Ability™ SafetyInsight™, https://new.abb.com/oil-and-gas/products/automation/safetyinsight, accessed: 06.05.2026
[19] Neo4j, "*GraphRAG*", https://neo4j.com/labs/genai-ecosystem/graphrag/, accessed: 29.04.2026
[20] Neo4j, https://neo4j.com/, accessed: 6.05.2026