AHEAD × neo4j

WHITEPAPER



DIG

The Declarative-Imperative Graph

A universal modelling methodology for the enterprise in the AI era

Ragnar Láki Jónsson

Co-founder & CTO, Ahead ehf. · November 2026

Abstract

An enterprise is represented across many capable systems, but rarely as one coherent model. Customers, products, people, contracts, regulations, controls, evidence, events and operational state are held in different structures with different semantics. Questions that cross those boundaries must therefore be reconstructed from several systems, and AI systems inherit the same fragmentation.

This paper introduces DIG, the Declarative-Imperative Graph: a universal methodology for representing enterprise reality in one graph model. DIG makes three moves. First, it brings declarative structure (what must, may or must not be true) together with imperative structure: the events, observations and proposed state changes through which the enterprise evolves. Second, it lifts the entities involved into high-dimensional objects, or citizens of the graph, with current state, history, relationships, behavioural expectations and governed lifecycles. Third, it makes the graph the authoritative substrate rather than a view reconstructed from another storage model.

The methodology defines a public primitive contract for citizens, declarations, events, trajectories, evidence, actors, authority, findings, sources, versions and graph state. It then applies that contract through four reasoning classes: hybrid conformance, cross-domain reasoning, enterprise AI reasoning and automated-actor supervision. Each class runs over observed graph state and over branched, hypothetical state. On a branch, the same evaluations determine admissible futures. Consequential automated action is governed through an explicit control boundary between DIG, which evaluates admissibility, and operational engines, which execute.

DIG's strongest properties are conditional on implementation. Within a faithfully modelled and declared scope, graph-derived findings can be traceable to their evidence paths, bounded by the graph state on which they depend, and replayable as the same canonical query against the same graph state and evaluation version. Missing, stale or insufficiently modelled knowledge is not treated as a positive verdict; it becomes an explicit condition that guides further modelling. Initial implementations of DIG are underway. This paper defines the target methodology and architectural contract; it does not present completed benchmarks or production outcome studies.

Scope, assurance and implementation conditions

DIG is a methodology rather than a guarantee supplied by a database or product. Its properties arise only where the relevant enterprise reality has been modelled carefully, declarations have been reviewed by competent domain owners, source systems are linked and governed, object lifecycles are maintained, authority and evidence requirements are explicit, and the operational execution surfaces in scope enforce DIG's decisions.

Authoritative scope

Universality does not mean that an enterprise must model everything before DIG becomes useful. It means that the same primitive contract can extend across domains without changing methodology. A particular implementation becomes authoritative incrementally and must declare the boundary within which its findings are intended to carry weight.

That boundary includes:

- The enterprise domains, citizen classes and instances covered.
- The declarations and versions represented.
- The sources connected and their accepted synchronisation age.
- The evidence types recognised.
- The authority relations and actor classes governed.
- The queries and evaluation versions in force.
- The execution surfaces that consult DIG before committing actions.
- Any exclusions, unresolved mappings or known limitations.

Outside this boundary, DIG makes no claim of completeness or authority. Inside it, incompleteness is itself part of the model.

Assurance is implementation-dependent

DIG does not make an incorrect source assertion true. It does not guarantee that a clause has been modelled faithfully, that an event was observed, or that an executor cannot bypass a control it has not integrated. It provides the structure through which source, version, scope, evidence, authority, review and execution can be made explicit and inspected together.

The properties used throughout this paper therefore have bounded meanings.

Property	Meaning in DIG
Traceability	A graph-derived finding can be reduced to an inspectable path through a specified graph state.
Reliability	A finding is bounded by the facts, declarations, evidence, authority and declared scope represented in that graph state.
Replayability	The same canonical query against the same graph state and evaluation version returns the same graph-derived verdict.

Traceability is not proof that every input was correct. Reliability is not universal truth. Replayability applies to the graph-derived verdict, not necessarily to the natural-language interpretation or wording used to present it.

Incomplete knowledge is visible and actionable

DIG does not turn missing knowledge into an affirmative conclusion. Depending on the implementation and domain, an evaluation can resolve to states such as *satisfied*, *violated*, *pending*, *not applicable*, *not evaluable*, *evidence insufficient*, *source stale*, *model incomplete* or *outside authoritative scope*.

The distinction between the last two conditions matters. *Outside authoritative scope* means DIG makes no claim. *Model incomplete* means the question falls within a declared scope but has exposed a modelling, evidence, synchronisation or governance gap that requires attention.

Operational use therefore creates a modelling feedback loop. Questions the graph cannot resolve reveal the next relationship to map, source to synchronise, declaration to review, identity to reconcile, evidence requirement to define or authority relation to govern. Once those additions have been reviewed and incorporated, the graph's authoritative coverage grows. In this limited but important sense, DIG helps grow itself: not by inventing enterprise meaning autonomously, but by turning the gaps in its representation into explicit, traceable and actionable conditions.

Public contract and implementation boundary

This paper publishes the semantic obligations and observable invariants of DIG, not the proprietary machinery by which an implementation satisfies them. A conformant implementation makes declarations typed, scoped and versioned; binds evaluations to graph states; produces findings from a declared outcome set; connects those findings to their citizens, declarations, evidence and authority; versions changes to evaluation behaviour; and supports replay against the same graph state and evaluation version.

The formal representations, evaluation algorithms, propagation machinery, query compilation and internal implementation architecture are outside the scope of this paper and form part of the implementation IP. Section 3.5 sets out which capabilities the Neo4j platform provides and which DIG implements, so that neither is credited with the other's work.

Implementation status

DIG is a newly defined methodology. Initial implementations are underway at the time of writing. This paper describes the target architecture and the properties expected of conformant implementations. It does not present completed performance benchmarks, comparative implementation results or production outcome studies. Implementation conformance, query replay, source freshness, control-boundary latency, propagation behaviour and operational case studies are subjects of ongoing work and future publication.

1. The enterprise has no single model

Walk into a large organisation and ask to see the enterprise. What appears is a tour of systems, each representing one slice of reality. The CRM holds customers and opportunities. The process mining platform reconstructs operational flows. Workflow engines move work forward. Regulatory libraries hold standards and clauses, often as prose. GRC platforms hold controls, evidence catalogues and findings. Master data systems maintain canonical identifiers. Document repositories hold policies and contracts. HR systems hold people, roles and certifications. Product systems hold parts, configurations and bills of material.

These systems are often competent and locally authoritative. Their shared limitation is that they model different things through different primitives.

Module class	Primary representation	Enterprise context normally outside it
Process mining	Observed flow reconstructed from event logs	Governing declarations and live object state across domains
Workflow engine	Prescribed and executing work	Full external context bearing on admissibility
CRM	Customers, accounts and commercial activity	Regulatory, control and operational dependencies
Regulatory library	Standards and clauses	Live applicability, object-level status and operational evidence
GRC platform	Controls, evidence and findings	Product, customer and operational trajectories
Master data	Canonical identity and reference attributes	Behaviour, lifecycle and governing declarations
Ontology or semantic model	Domain structure and vocabulary	Live observations, state changes and execution authority
Document repository	Authoritative text and artefacts	Executable meaning and evidence bindings
HR system	People, roles and certifications	Runtime authority in the full context of an action
Product data system	Parts, configurations and specifications	Operational trajectory and state in use

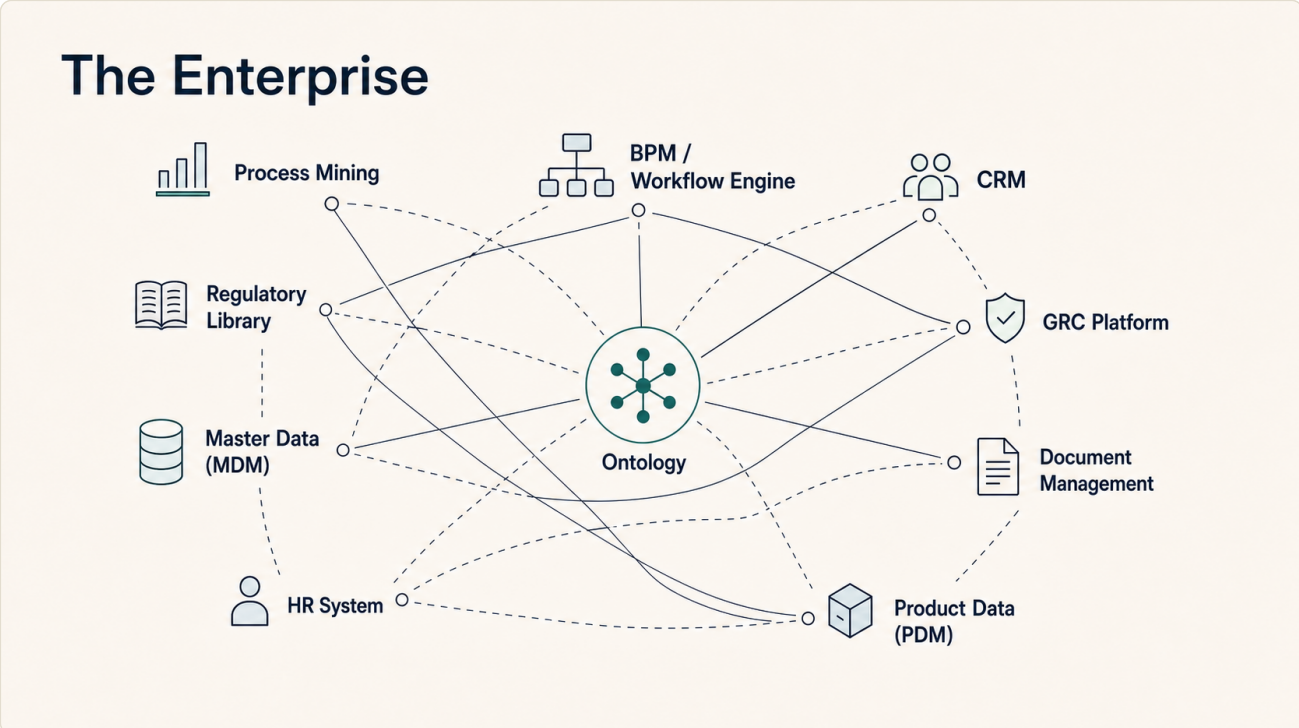


Figure 1. Enterprise meaning is distributed across capable systems and reconstructed through integrations.

The gap becomes visible whenever a question crosses systems. Consider:

> *Is this customer in good standing, given the commitments in its contract, the regulations applying to its jurisdiction, the controls operating on its data, the evidence currently available, and the live processes running on its account?*

No individual module is expected to answer this. The answer is assembled by fetching partial representations and reconciling them. The same occurs in product release, procurement, access review, safety, audit, finance and regulatory change. The enterprise has information about itself, but the connected meaning is reconstructed when needed rather than maintained continuously.

AI systems absorb this gap sharply. A model can call tools, retrieve documents and combine responses, but tool access does not manufacture missing relationships or common semantics. If the obligation, event, citizen state, evidence and authority were never represented as one connected structure, an AI answer that joins them is a reconstruction. Better retrieval can improve the reconstruction; it cannot make the underlying substrate coherent after the fact.

Systems of record and operational engines

DIG relates differently to two broad parts of the estate.

Systems of record hold authoritative content about a particular domain. During adoption, DIG can coexist with them, synchronise their content and make its own authority explicit by domain. In some cases the external system remains authoritative. In others, reviewed graph structure may gradually become the authoritative representation for the domain. That migration is an organisational and governance decision, not an automatic consequence of deploying a graph.

Operational engines do something in the world. They send messages, post transactions, grant access, execute workflows, sign documents, call APIs or actuate equipment. DIG does not replace these engines. In the target architecture, engines governing consequential actions consult DIG before committing them. DIG evaluates admissibility; operational engines enforce execution.

A running example: Acme Wings

Consider a fictional maintenance and repair organisation, Acme Wings MRO. Its maintenance system records work orders, its inspection application holds evidence, its HR system maintains technician certifications, its security platform detects intrusions, and its regulatory library holds airworthiness requirements. Each system is locally useful.

If a security intrusion makes maintenance records from a particular time window untrustworthy, the operational consequence crosses every one of those domains. The organisation must find the affected records, identify the work orders and aircraft that depend on them, determine which regulatory evidence conditions are no longer satisfied, decide which aircraft states are admissible, and govern the actions of human and automated responders.

Before DIG, that consequence is assembled through coordination. With DIG, the intrusion can be represented as a state change on the affected system citizen, connected to the records, work orders, aircraft, evidence requirements, actors and declarations that depend on it. The rest of this paper returns to this example as each part of the methodology is introduced.

2. The DIG model

DIG begins from three observations. Enterprise reality contains both normative and observed structure. The entities affected by that structure have state and lifecycles of their own. The relationships joining these elements carry meaning that must survive storage, query, change and replay.

DIG addresses these through three moves, in one canonical order:

1. Unite declarative and imperative structure.
2. Lift enterprise entities into stateful, high-dimensional citizens.
3. Make the graph the authoritative substrate in which both reside.

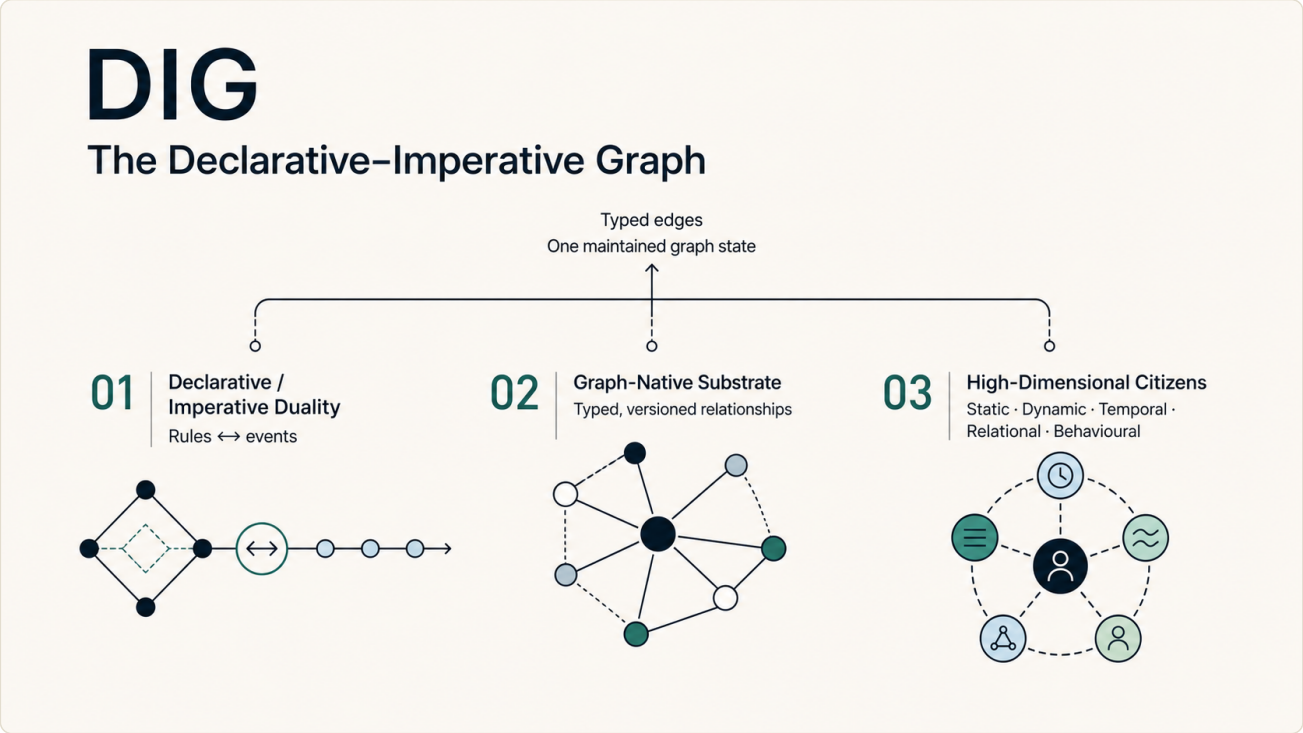


Figure 2. The three DIG moves.

2.1 Declarative and imperative structure

An enterprise contains statements about what must, may or must not be true, and records of what is, was or is proposed to become true. DIG calls the first *declarative* and the second *imperative*.

The terminology is broader than its use in programming languages. It is also distinct from the narrower contrast between declarative and imperative process models. Here, *declarative*

refers to normative structure and *imperative* to the typed events, observations and proposed transitions through which enterprise state evolves.

Domain	Declarative structure	Imperative structure
Regulatory	Clauses, obligations, exclusions and evidence requirements	Evidence, violations, submissions and correspondence
Commercial	Contractual commitments, SLAs and entitlements	Account activity, payments and commitments met or missed
Product	Bill of materials, quality specification and release rule	Assembly, inspection, deviation and release events
Control and GRC	Control definition, frequency and evidence requirement	Executions, exceptions, attestations and audit events
Identity and HR	Entitlement, delegation and certification requirements	Grants, approvals, logins, training and revocations
Safety and operations	Operating envelope and maintenance obligations	Measurements, deferrals, findings and state transitions

The two structures are complementary. A regulation with no connected operational evidence is a prescription whose status cannot be evaluated. An event stream with no applicable declarations can describe activity but not whether it was required, permitted or prohibited. Useful enterprise determinations depend on the intersection.

Earlier work in declarative process modelling, declarative process mining and hybrid process discovery has brought constraint-based and flow-based descriptions together within processes [2, 3]. Object-centric extensions broaden the units participating in those processes [1]. DIG adopts important insights from these traditions but changes the scope and factoring. Regulations, contracts, entitlements, product specifications, controls, authority and evidence are not treated as adjuncts to one process model. They are co-resident graph structures that can bind to citizens and events across domains.

In DIG, a declaration is typed, scoped and versioned. It may express an obligation, permission, prohibition, condition, exclusion, entitlement, applicability relation or evidence requirement. An event is a typed observation or proposed change connected to the citizens it concerns, the actors involved, its time and source, its evidence, and the declarations that bear on it.

Because the two structures share the graph, traversal can proceed in either direction: from an event to the declarations that govern it, or from a declaration to the citizens, trajectories, evidence and findings connected to it. Declaration and observation continuously inform one another through the maintained graph state.

2.2 High-dimensional citizens

A declaration says what should be true. An event says that something happened or is proposed. Neither alone answers the ordinary operational question: what is the current state of the affected entity?

A customer may be active, under review, in dispute or restricted. A contract may be pending, executing, breached or terminated. A regulation may be in draft, in force or superseded. An aircraft may be in service, under maintenance or grounded. A technician may have an active, expiring or suspended certification. These states are not incidental attributes of whichever event last mentioned the entity. They are maintained positions in governed lifecycles.

DIG calls an entity modelled in this way a *high-dimensional object*, or a *citizen of the graph*. A citizen is a first-class graph entity that bears state, participates in typed relationships, carries history, is bound to applicable declarations and evolves through a governed lifecycle.

Every citizen is described through five core modelling dimensions.

Dimension	Core question	Typical content
Static	What is it?	Identity, type, classification, ownership and jurisdiction
Dynamic	What is it doing now?	Lifecycle state, current measurements and operational status
Temporal	Where has it been?	Prior states, event participation, revisions and historical metrics
Relational	How is it connected?	Dependencies, governance, composition, evidence and authority
Behavioural	What is expected of it?	Baselines, ranges, cadence and anomaly signatures

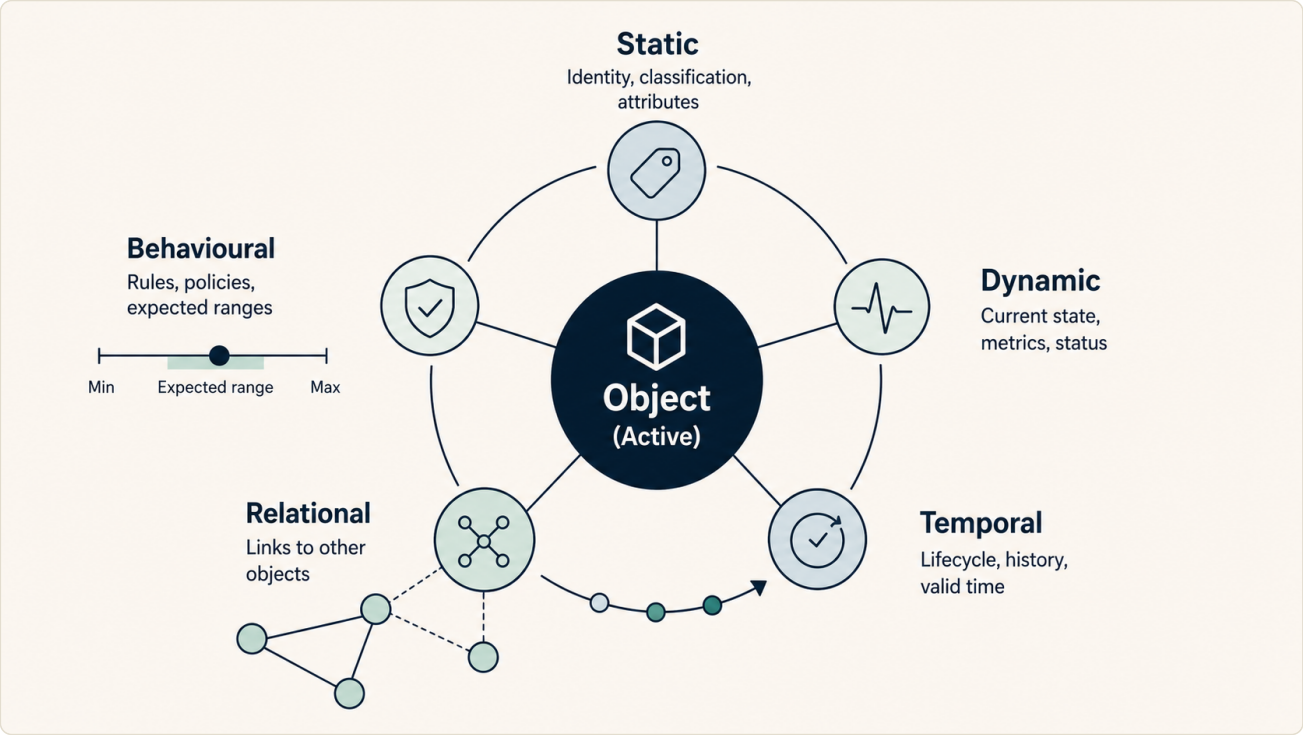


Figure 3. A high-dimensional citizen combines identity, live state, history, relationships and behavioural expectations.

The five dimensions are a modelling contract, not a claim of mathematical uniqueness. They ensure that DIG asks the same five questions of every citizen even when a particular dimension is sparse for a particular type. Domain models can add provenance, access, valuation, intent or other dimensions without changing the common contract.

The result is comparable to a network of micro twins. Each citizen is a faithful, queryable representation of one enterprise entity within the scope and fidelity declared for it. A customer, contract, regulation, product, control, person, organisational unit, risk, system and aircraft can all be citizens while retaining distinct domain-specific types, declarations and lifecycles. Common primitives do not erase those differences; they allow the differences to participate in common reasoning.

2.3 Governed lifecycles and trajectories

Each citizen has an explicit lifecycle appropriate to its type. Events propose or record transitions. Declarations determine whether the transition is required, permitted, prohibited, pending further evidence or outside the current scope of evaluation.

A transition can therefore be asked as an admissibility question:

> *Given this citizen's current state, the declarations now applicable, the evidence present, the actor and authority involved, and the graph state being evaluated, may this transition occur?*

The public contract requires the inputs and resulting finding to be explicit. The proprietary evaluation machinery by which an implementation resolves the result is not specified here.

A *trajectory* is the connected subgraph of events through which an enterprise activity unfolds. A process instance is one kind of trajectory, but the concept also covers a contract lifecycle, product build, audit period, customer engagement, incident response, regulatory enactment or maintenance episode. Events in a trajectory connect to the citizens they concern, the actors involved, the evidence produced and the declarations active at the time.

This makes current state and historical path available together. The event history explains how a citizen arrived where it is; the maintained citizen state makes the current answer directly queryable; and the active declarations determine which next transitions are admissible.

In Acme Wings, the maintenance system is itself a citizen. The security intrusion changes its integrity state. Work orders recorded through the system are connected trajectories; inspection records are evidence; aircraft and technicians are citizens; airworthiness clauses and certification requirements are declarations. The incident matters operationally because those elements are already connected, not because a report later guesses how they relate.

2.4 The same lenses across domains

The five dimensions are deliberately domain-independent; one substrate serves any vertical.

Citizen type	Static	Dynamic	Temporal	Relational	Behavioural
Customer	Identity, segment, jurisdiction	Standing and current restrictions	Interactions and commitments met or missed	Contracts, entitlements, deals and controls	Expected engagement and transaction envelope

Citizen type	Static	Dynamic	Temporal	Relational	Behavioural
Contract	Parties, subject and jurisdiction	Execution, breach or termination state	Amendments, exercises and prior states	Obligations, governed services and regulations	Expected lifecycle and performance cadence
Regulation	Source, clause structure and version	Draft, in force or superseded	Enactment and supersession history	Jurisdictions, citizen classes and derived controls	Expected review or implementation pattern where relevant
Control	Definition, owner and frequency	Effectiveness and overdue state	Executions, failures and evidence history	Risks, standards, processes and evidence	Expected execution and evidence cadence
Aircraft	Registration, type and configuration	Airworthiness, location and maintenance state	Flights, inspections, maintenance and deviations	Components, operators, work orders and regulations	Dispatch, reliability and maintenance envelope
Person	Identity, role and organisational unit	Active assignments and certifications	Training, delegation and work history	Reporting lines, authority, tasks and assets	Expected workload and qualification pattern

The behavioural dimension does not imply that every citizen has intention or agency. It records the expected envelope against which divergence is meaningful. For a person or customer, that may describe activity patterns. For a control, it may describe evidence cadence. For a regulation, the useful behavioural content may be limited to expected review, enactment or implementation patterns. Where a dimension is not meaningful to a specific subtype, the model should say so rather than inventing content to fill it.

The common lens makes cross-domain questions possible without flattening domain distinctions. A technician certification and an aircraft airworthiness state remain different concepts with different owners and declarations. Their common citizen contract allows a query to traverse the authority and maintenance relationships between them while preserving those types.

3. Primitive and substrate contract

The same primitives must support every domain represented in DIG. Domain vocabularies remain specific, but each domain maps to a stable graph-level contract.

3.1 Minimum primitive contract

Primitive	Public definition
Citizen	An enterprise object bearing maintained state and described through static, dynamic, temporal, relational and behavioural dimensions.
Declaration	A typed, scoped and versioned normative graph element expressing an obligation, permission, prohibition, condition, exclusion, entitlement, applicability relation or evidence requirement.
Event or action	A typed observation or proposed state change bound to an actor, one or more citizens, time, source, applicable declarations and evidence.
Trajectory	A connected subgraph of events representing an unfolding enterprise activity.

Primitive	Public definition
Evidence	An artefact, observation, attestation, signature, measurement, log entry or source reference supporting a finding or transition.
Actor	A human, AI agent, workflow bot, integration service, operational engine, external system or organisational role capable of proposing, approving, executing or reviewing events.
Authority	A typed relation determining who may propose, approve, reject, delegate, execute, reverse or review an action under specified conditions.
Finding	A graph-resident result concerning a declaration, citizen, trajectory, actor or proposed action in a specified graph state.
Source	The system, document, standard, sensor, service, person or process from which a graph assertion originated.
Version	The identity of an element or evaluation definition at a particular point in its governed history.
Graph state	The versioned boundary of citizens, declarations, relationships, trajectories, evidence, authority, sources, synchronisation status and evaluation definitions used for a query or decision.

The primitive contract is deliberately small. It lets an aircraft, contract, person and regulation carry different domain models while participating in the same high-level queries. It also makes scope visible. A finding can state not only what was concluded, but which versions, sources and relationships were inside the graph state from which the conclusion followed.

Relationships are typed facts. Where a relationship itself requires identity, history, declarations, evidence or authority, it is represented in a form that can carry those properties explicitly rather than being left as an unversioned connector. The methodology requires the meaning to remain addressable, which calls for a substrate in which relationships are first-class and carry state of their own. A labelled property graph provides this directly: in Neo4j, every relationship is typed, stored, directly traversable and able to carry its own properties.

3.2 Graph-native architecture and traversal performance

DIG requires the graph to be the authoritative semantic form rather than a display assembled from application conventions at query time.

The DIG substrate has the following observable properties:

- Typed relationships are persisted, queryable and versioned.
- Relationship identity, provenance, scope and temporal validity survive traversal.
- Heterogeneous citizens, declarations, events, evidence, actors and authority coexist under one graph-level query model.
- Paths are inspectable results, rather than intermediate joins discarded after a report is produced.
- A graph state can be pinned for decision, evidence and replay.
- The same primitive contract operates across every domain represented.

A native graph database is the natural physical implementation because it stores nodes and relationships as primary structures and executes traversal directly over them. Ahead's implementation runs on Neo4j (Section 3.5).

What matters is where the semantic burden resides. If the meaning of *governs*, *depends on*, *satisfies*, *delegated to* or *derived from* exists only in application code that reconstructs joins for each use case, then the authoritative model remains outside the data substrate. DIG requires those meanings, their versions and their evidence paths to reside in the maintained graph state.

The choice of substrate also determines how DIG performs, because DIG's central workloads are traversals. Cross-domain propagation (Section 4.2), evidence-path assembly (Section 5.4) and the unscoped diff of a branch (Section 6.4) each follow typed dependencies outward from a changed fact to a depth that is unknown before the query runs. A propagation from a changed declaration version may end after two hops or continue through ten, across citizen types no single application schema anticipated.

In a join-based store, each hop is a join or index lookup whose cost grows with the size of the data involved, and variable depth requires recursive query constructs that are hard to plan and optimise. Native graph storage with index-free adjacency gives each node direct references to its relationships. A hop follows a stored reference, so the cost of a traversal tracks the part of the graph it touches and stays largely independent of the total size of the graph [12]. Neo4j pairs that storage with Cypher. Its quantified path patterns, whose syntax follows the ISO GQL standard [13], express variable-depth traversal declaratively, apply filters during expansion and return paths as first-class values, which meets DIG's requirement that paths be inspectable results.

Performance still depends on topology, cardinality, indexing, distribution and query design, and relational systems serve many shallow, fixed-depth graph queries well. DIG's workloads vary in depth and cross domains by construction. Native graph storage and processing is designed for exactly this class of workload, and it performs best there. DIG's substrate choice therefore rests on two grounds: connected meaning must be preserved uniformly, and the traversals that carry that meaning must stay fast as the graph grows across domains.

3.3 Graph state, time and provenance

Replay requires more than an event timestamp. A DIG graph state identifies the versions of the facts and semantics on which a determination depends. It includes declaration versions, citizen states, relationship versions, evidence, authority, actor state, source synchronisation status and the evaluation definitions applied.

This matters when reality is corrected. Sources can send late events. Identities can be reconciled. Evidence can be withdrawn. A contract interpretation can be revised. A regulation can be superseded. DIG does not require history to pretend that the earlier graph state was correct; it requires the earlier state, the correction and the governed reason for the correction to remain distinguishable.

Provenance standards such as W3C PROV-O demonstrate a general vocabulary for relating entities, activities and responsible agents across systems [7]. DIG's provenance obligations are part of a larger operational contract: source and derivation are connected to the declarations, state transitions, authority and findings that depend on them.

3.4 Observable semantics without disclosed machinery

Different domains may require different declaration formalisms. DIG does not require the internal expression of an airworthiness rule to be identical to that of a commercial entitlement. It requires a common graph-level contract.

Every declaration is typed, scoped and versioned. Every evaluation identifies its graph state and evaluation version. Every resulting finding uses a declared outcome vocabulary and exposes the evidence path on which it depends. Conflicts, exceptions, precedence and temporal conditions are resolved through governed semantics whose versions are part of replay.

These invariants are public and testable. The algorithms and representations that fulfil them are not disclosed in this paper.

3.5 Implementation on Neo4j

Ahead implements DIG on Neo4j, and each has a distinct role. Neo4j provides the native graph platform: storage, traversal, query, transactions, scale and data-level security. DIG provides the enterprise semantics that run on it: the primitive contract, declarations and their evaluation, graph state and replay, branching, authority and the control boundary. The table below sets out the division capability by capability.

Capability	Provided by Neo4j	Defined and implemented by DIG
Graph structure	Native property-graph storage of labelled nodes and typed relationships, each carrying properties, with index-free adjacency	The primitive contract: citizens, declarations, events, trajectories, evidence, actors, authority, findings, sources, versions and graph state, and the obligations each carries
Traversal and query	Cypher pattern matching with variable-length and quantified path patterns, shortest paths and paths returned as values	Canonical queries, propagation semantics, evidence-path construction and the outcome vocabulary of findings
Time and replay	Temporal value types and transactional commit ordering	Graph states, pinning, replay against the same graph state and evaluation version, and distinguishable correction history
Branching	The same storage, traversal and query engine applied to branch state	Branch semantics, isolation of hypothetical state from live facts, evidence grades and the branch diff
Data access	Role-based privileges on databases, labels, relationship types and properties, including property-based rules	Authority relations, oversight modes, separation of duties and relationship- and attribute-based refinement by citizen state, purpose and context

The division has a practical consequence for assurance. Whether a finding is correct depends on DIG's semantics and on the modelling, evidence and governance described in the scope section. How quickly it is reached, how consistently it is written and how its underlying data is protected depend on Neo4j. Neither layer substitutes for the other.

4. Live reasoning

With declarations, trajectories, citizens and evidence co-resident, DIG supports repeatable classes of evaluation. A reasoning class is not a product module. It is a pattern of questions made possible by the common substrate.

Class	Core question
I. Hybrid conformance	Is this subject admissible against its declarations, actual trajectory and citizen states?
II. Cross-domain reasoning	What else becomes affected when this fact, declaration, state or relationship changes?
III. Enterprise AI reasoning	Which conclusions and recommendations can be grounded in replayable graph findings?
IV. Automated-actor supervision	What may this actor do under the current evidence, authority and oversight conditions?

The classes are state-invariant. Each runs over observed graph state and over branched, hypothetical state, where the same evaluations determine admissible futures. This section defines the four classes as they run over observed state. Section 5 develops the control architecture through which automated-actor supervision governs consequential action. Section 6 develops the branch model through which all four classes run over hypothetical state.

4.1 Reasoning Class I: Hybrid conformance

Hybrid conformance evaluates a bounded subject: a customer engagement, contract lifecycle, product build, control period, audit, incident, work order or other trajectory. It asks whether the subject is conformant when three assessments are made together.

Assessment	Question	What it prevents from disappearing
Declarative	Which applicable declarations are satisfied, violated, pending or otherwise unresolved?	Normative obligations and evidence requirements
Imperative	How does the observed trajectory relate to the reference or approved behaviour for this kind of subject?	Flow, sequence, timing and trajectory anomalies
Object	Are the involved citizens in states that permit the activity or transition?	State-dependent admissibility outside the local flow

The word *score* may be used for these assessments, but it does not require that every domain reduce them to a single number. A score can be categorical, ordinal or quantitative according to the declared implementation. What matters is that the three facets remain separately inspectable before any combined verdict is presented.

They can disagree. A transaction may follow an approved trajectory and meet its commercial rules while remaining inadmissible because the customer has entered sanctions review. An audit may follow every expected step while evidence arrived after a declared deadline. A manufacturing build may deviate from its common variant while satisfying every specification and leaving all involved objects in valid states.

Treating citizen state independently is important. If object state is reconstructed only as a side effect of replaying activity history, its current admissibility and the provenance of that state remain entangled with the trace. DIG maintains the citizen state and the trajectory together, allowing the disagreement to be surfaced as part of the finding.

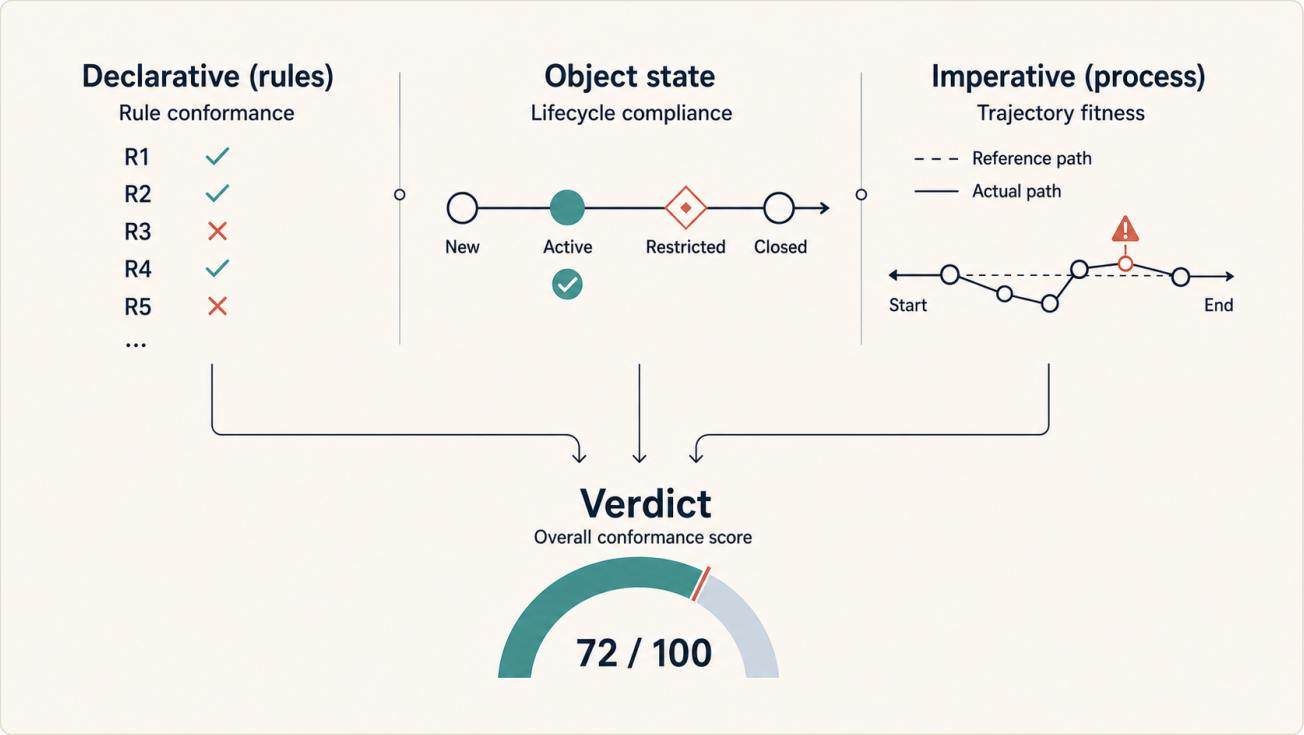


Figure 4. Hybrid conformance keeps declarative, object-state and imperative assessments separately inspectable before a verdict is presented.

In Acme Wings, an inspection trajectory can be complete and perfectly ordinary while the maintenance system citizen has moved into a compromised integrity state. The trajectory assessment holds, while the object and declaration assessments fail. Hybrid conformance preserves that distinction.

4.2 Reasoning Class II: Cross-domain reasoning

Cross-domain reasoning begins where the local verdict ends. Its core question is:

> *What else becomes satisfied, violated, pending, at risk, stale or not evaluable when this graph fact changes?*

A change can be an event, declaration version, citizen transition, evidence update, source-status change or relationship change. The graph identifies the connected citizens, trajectories, declarations and evidence paths whose meaning depends on it. Those affected elements are then evaluated under their current scope and semantics, and the resulting findings are written back with their evidence paths.

The architectural pattern is:

1. Record the change as a versioned graph fact.
2. Identify the connected elements whose meaning depends on that fact.
3. Evaluate the affected subjects under the declarations and graph state now applicable.
4. Record findings, obligations, holds, evidence gaps or proposed transitions.

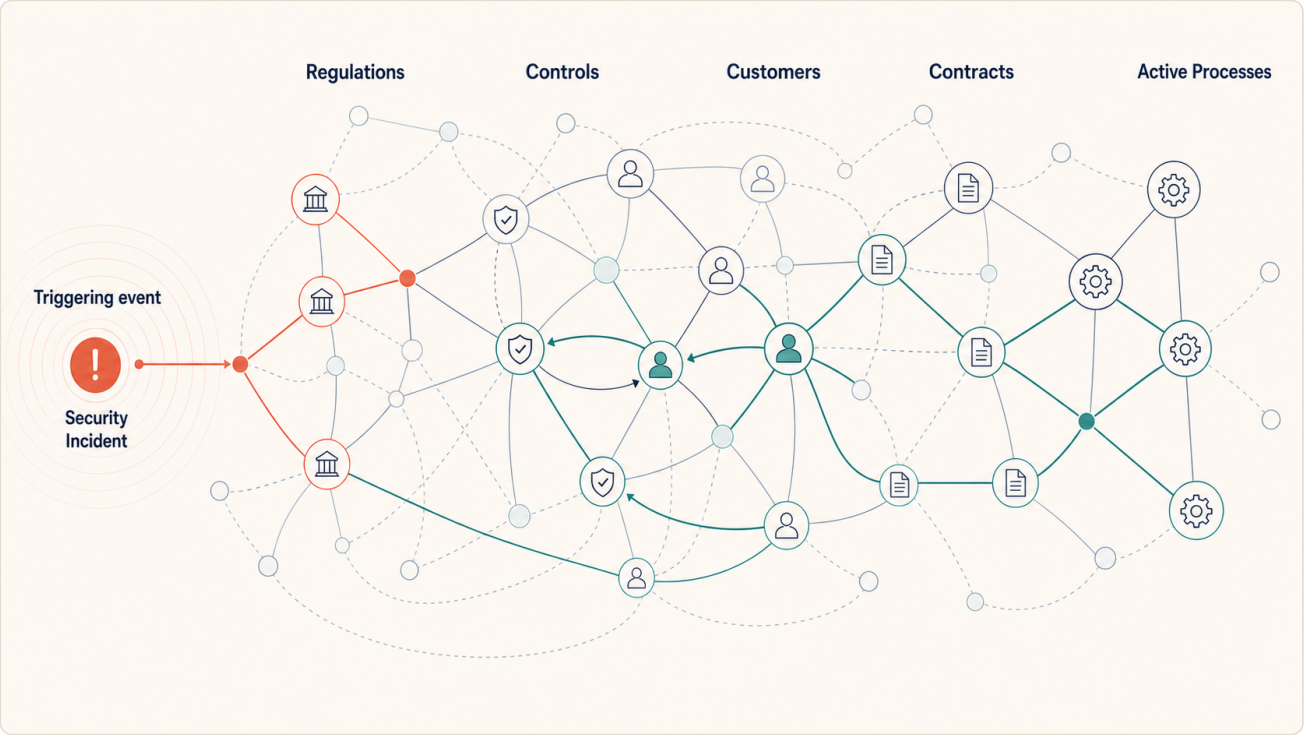


Figure 5. A triggering event propagates only through represented dependencies, exposing affected subjects and modelling gaps.

This turns several familiar enterprise tasks into forms of graph reasoning. Regulatory impact analysis becomes a traversal from a changed declaration version. Evidence assembly becomes a path query. A customer view becomes a rooted subgraph across contracts, controls, regulations and live engagements. A certification expiry exposes every task, asset and approval that depends on it. Each is a traversal whose depth follows the represented dependencies, the workload Section 3.2 identifies as the strength of native graph processing.

The methodology does not claim that every consequence is intrinsically easy. It claims that dependencies already represented in the graph do not need to be rediscovered through a new integration exercise for each question. Missing dependencies instead produce explicit modelling gaps that feed the growth loop defined in the scope section.

4.3 Reasoning Class III: Enterprise AI reasoning

Connecting an AI model to a graph does not make the model inherently reliable. DIG addresses the surface on which enterprise claims are resolved.

An AI system can use DIG to obtain graph-derived findings whose facts, declarations, citizen states, evidence, sources and authority are explicit. The model may explain those findings, compare alternatives, draft communication or recommend action. The enterprise determination carrying the load remains the graph result.

This produces three separations.

First, graph grounding is distinguished from language generation. A model may vary its wording without changing the underlying finding.

Second, retrieval is distinguished from authority. A retrieved document may be relevant, but a DIG declaration states whether its represented content is active, scoped and authoritative for the question.

Third, a natural-language question is distinguished from the canonical graph query into which it is resolved. The same wording may be interpreted differently by an LLM. Replayability begins after the interpretation has been fixed: the same canonical query, graph state and evaluation version return the same graph-derived verdict.

An AI answer can therefore contain two kinds of claim. Graph-grounded claims are tied to findings and evidence paths. Generative commentary (analogy, summary, tone, comparison or suggestion) remains model output and should not be represented as if it were a deterministic graph fact.

For an auditor, the useful question is no longer whether the AI can reproduce an internal chain of thought. It is which findings, declarations, citizen states, sources and evidence the answer relied upon. That path remains inspectable independently of the language model that presented it.

The same scope boundary applies. If a required domain is outside DIG’s authority or a source is stale, the AI system should receive that condition as part of the result. DIG's value is that it makes visible where grounded determination ends and unsupported synthesis begins.

4.4 Reasoning Class IV: Automated-actor supervision

Enterprise AI reasoning concerns what a system can conclude or recommend. Automated-actor supervision concerns what a human, AI agent, workflow bot, integration service or operational engine may do.

The distinction is essential. A model can draft freely while remaining unable to commit a consequential state change. Conversely, a deterministic integration service can carry broad operational authority despite having no generative model. DIG governs actors through their relationship to actions, citizens, declarations, evidence and authority rather than through a label such as AI.

Supervision is a reasoning class because its verdict is a graph finding like any other. The evaluation binds the actor, the proposed action, the affected citizens, the applicable declaration versions, the evidence present and the authority invoked, and it resolves to an oversight mode rather than a bare permit or deny. Section 5 develops the machinery this class requires: the action ontology, the authority and oversight relations, and the control contract through which the verdict governs consequential execution.

4.5 The reasoning sequence

The classes form a sequence, but each produces a graph-resident result independently.

1. Hybrid conformance resolves the local subject across declaration, trajectory and citizen state.

2. Cross-domain reasoning identifies the connected subjects whose meaning may have changed.
3. Enterprise AI reasoning exposes those findings for explanation, comparison, recommendation or interaction.
4. Automated-actor supervision determines, where a finding meets a proposed action, what an actor may now do.

The sequence is not a licence for AI to fill missing graph structure. When a local evaluation is not possible, the result remains *not evaluable*, *source stale*, *evidence insufficient* or *model incomplete*. Cross-domain reasoning can propagate the consequence of that uncertainty, and an AI system can explain it, but neither converts it into a fact.

This is also where the modelling feedback loop enters daily operation. A repeated class of unevaluable questions can be grouped into a governed modelling backlog. Domain owners can decide whether the missing relationship or declaration belongs inside the authoritative scope, review the proposed representation and activate it through the same version and evidence disciplines applied elsewhere. The graph grows through governed correction, not through an AI silently adding whatever relation would make an answer possible.

5. Governed automated action

Reasoning Class IV renders verdicts about actors and their proposed actions. This section defines the architecture that makes those verdicts binding: the ontology that types an action, the authority and oversight relations that scope it, and the control contract through which an admissibility verdict becomes the condition of consequential execution.

5.1 Action ontology

Every proposed action belongs to a typed hierarchy.

Level	Meaning	Example
Domain	Enterprise area in which the action occurs	Customer, finance, product, identity, safety
Trajectory	Unfolding activity to which it belongs	Complaint, access review, maintenance deferral
Decision type	Class of decision or transition	Send communication, grant access, release asset
Action instance	Concrete proposal at a particular time	Send this message now; release this aircraft under this evidence

The hierarchy permits inheritance. An action instance can inherit applicable risk attributes, declaration bindings, evidence requirements, authority and oversight posture from its decision type, trajectory and domain, with governed overrides where the model permits them.

The action is therefore not evaluated as raw text or an isolated API call. It is a proposed event in a known part of the enterprise model.

5.2 Permission, authority and oversight

Permission answers whether an action is admissible under the declarations and citizen states that govern it. Authority answers who is competent to exercise judgment over that action. An action can be generally permitted but still require an accountable human approval. It can be permitted for one actor class and prohibited for another. It can be executable only below a threshold or only when specified evidence is present.

Authority relations can express capabilities such as *may propose*, *may draft*, *may approve*, *may execute*, *may delegate*, *must review after* and *may reverse or remediate*. Their scope includes the actor, action type, citizen class or instance, declaration set, time and oversight mode.

DIG supports more than a binary choice between autonomy and human approval.

Oversight mode	Meaning
Blocked	The action cannot be proposed or executed in the current graph state.
Draft only	The actor may prepare content or a proposed transition but cannot commit it.
Human approval	A competent authority must approve before execution.
Supervised execution	The action may proceed under declared monitoring and interruption conditions.
Review after execution	The action may proceed but is routed for review within a declared window.
Autonomous	The action may execute without human intervention under the current graph state.

Oversight belongs to the action instance, not permanently to the actor. A payment, customer message or maintenance decision can occupy different modes depending on amount, reversibility, citizen state, evidence, risk and authority.

5.3 The DIG control contract

Reasoning becomes operational only where executors enforce a control boundary. DIG evaluates admissibility; operational engines execute. The paper does not prescribe a single transport, transaction or orchestration protocol. It requires the following high level contract between DIG and every governed executor.

1. **Pre-commit evaluation.** A consequential action is evaluated before the executor commits it.
2. **Decision binding.** The verdict is bound to the exact actor, action, affected citizens, evidence, authority and graph state.
3. **Freshness and revalidation.** An expired decision, or one that no longer matches the relevant graph state, must be evaluated again.
4. **Executor acknowledgement.** Execution, blocking, failure, partial completion, reversal and remediation return to the graph as events with evidence.
5. **Declared failure posture.** Each action class defines whether loss of the DIG decision surface causes blocking, human approval, restricted operation or an authorised emergency path.

6. **Recovery and override governance.** Compensation, reversal and break-glass execution require explicit authority and leave evidence paths of their own.

This contract states the required architectural behaviour without disclosing the internal mechanism that binds, evaluates or distributes decisions. Particular implementations can use different transaction, gateway, workflow and integration patterns while remaining subject to the same public obligations.

Partial deployments can use DIG after execution as an evaluation and evidence layer. The strongest control claims in this paper apply only to action classes whose executors have implemented the pre-commit contract.

5.4 Evidence path and operational drift

Every proposed, approved, blocked, executed, failed, reversed or reviewed action leaves a versioned evidence path. The path identifies the actor, action type, affected citizens, applicable declaration versions, authority, oversight mode, graph state, evidence used, source systems involved and resulting transitions.

Operational drift is defined against that structure. It is not a change in the model's tone or style. It occurs when an actor's proposed actions, tool use, evidence behaviour, memory scope or authority claims diverge from its declared operating envelope.

An action outside the envelope is not accepted as the actor's newly discovered capability. It becomes a finding. Depending on the applicable declaration, DIG can block the action, require evidence, route it for approval, revoke or narrow authority, or move the actor into a stricter oversight mode.

Aggregate behaviour can also be evaluated. A changing action mix, rising override rate, repeated evidence failure, unusual tool sequence or persistent operation near declaration boundaries may change the actor citizen's state and reduce its autonomy. The graph does not attempt to stop a model from generating a novel plan. It prevents novelty from becoming consequential enterprise state change unless the resulting actions remain admissible.

5.5 Governing DIG itself

The graph that governs enterprise actors must itself be governed. Changes to declarations, authority, evidence, sources and evaluation versions are consequential events subject to the same disciplines of identity, access, approval, evidence and review.

Control objective	High-level posture
Identity and access	Role-based access establishes organisational roles; attribute- and relationship-based controls refine access by citizen state, domain, purpose, authority and context.
Least privilege	Actors and integrations receive only the graph permissions, action classes and tools required for their declared function.
Separation of duties	Material declaration, authority and execution changes separate proposing, approving and executing roles.
Change governance	Declarations, authority edges and evaluation versions follow reviewed activation, rollback and appeal lifecycles.

Control objective	High-level posture
Integrity	Material findings, approvals, transitions and evidence paths can be digitally signed or cryptographically chained to make alteration detectable.
Provenance	Assertions identify their originating source, time, connector or actor and the version under which they entered the graph.
Confidentiality and privacy	Access can be constrained at citizen, relationship, subgraph, property and evidence level, with minimisation, retention and correction governed by the applicable regime.
Resilience	Action classes declare recovery requirements and fail-closed, degraded or emergency operation.
Monitoring	Privileged changes, unusual traversals, evidence failures and repeated authority violations produce findings of their own.

RBAC provides a useful organisational baseline, while attribute-based access control incorporates subject, object, operation and environmental attributes into the decision [8]. DIG's connected model also makes relationship context relevant. These mechanisms complement rather than replace the declarations and authority relations used for operational actions. In Ahead's implementation, Neo4j enforces data-level access through role-based privileges on labels, relationship types and properties, including property-based rules. DIG's authority relations and oversight modes govern proposed actions above that layer (Section 3.5).

The posture is compatible with zero-trust principles in which identity and authorisation are evaluated for protected resources rather than inferred from network location [9]. DIG does not claim to be a complete zero-trust architecture; it supplies enterprise context that such enforcement can consult.

Recent analysis of agentic AI under European law similarly argues that governance begins with an inventory of an agent's external actions, data flows, connected systems and affected persons rather than only its internal model architecture [5]. DIG generalises that operational view: the graph governs the action attempted, the citizens affected, the authority invoked and the evidence required, whether the actor is generative, deterministic or human.

A cryptographic audit trail makes later alteration detectable. It does not make the original assertion true, and it must not be confused with universal immutability. Corrections, privacy obligations and governed deletion may require content to change while retaining an appropriate tombstone, signature or record of the authorised change.

Break-glass operation remains possible where a domain requires it, but only as a narrow and explicit authority path with defined duration, evidence and retrospective review. A safety architecture that cannot describe emergency operation has hidden rather than removed the exception.

5.6 The same control pattern across domains

The control machinery is stable even when domain vocabularies differ.

Domain	Representative citizens	Representative declarations	Governed actions
Customer	Customer, contract, entitlement, complaint	Communication, dispute, jurisdiction and refund rules	Draft or send communication, issue refund, change standing

Domain	Representative citizens	Representative declarations	Governed actions
Product	Product, component, configuration, release	BOM, quality, release and deviation rules	Release, hold, rework or approve deviation
Identity and HR	Person, role, entitlement, certification	Access, segregation-of-duties, delegation and training rules	Grant access, assign work, delegate or revoke authority
Finance	Invoice, vendor, payment, budget	Approval threshold, budget, payment and fraud rules	Route invoice, approve payment or hold transaction
Control and GRC	Control, risk, obligation, evidence	Frequency, ownership, sufficiency and closure rules	Request evidence, open finding, attest or close finding
Safety and operations	Asset, work order, technician, incident	Operating envelope, maintenance, qualification and evidence-integrity rules	Ground, release, defer or request reinspection

A specific deployment may govern only one row or even one action class within a row. The universal methodology does not imply that every executor is inside the boundary. Control claims must remain aligned with the declared integration scope.

6. Admissible futures

The four reasoning classes have so far operated over observed graph state. Admissible futures are what the same classes determine when they run over a graph state explicitly marked as hypothetical.

DIG is not a standalone simulation engine. It is a branchable operational substrate on which deterministic counterfactuals and generated simulations can run without rebuilding the enterprise's citizens, declarations, authority and dependencies in a separate structural model.

The key separation is between the authoritative structural model and the source of future events:

> *A future generator proposes events; DIG adjudicates and propagates them.*

A future generator can be as simple as one human-proposed action or as complex as a stochastic, optimisation or actor-based model. DIG does not replace those disciplines. It provides the branched enterprise state against which their proposed events are evaluated.

Class	On a branched graph state
I. Hybrid conformance	What the hypothetical trajectory makes required, forbidden or violated
II. Cross-domain reasoning	Where the hypothetical change propagates through represented dependencies
III. Enterprise AI reasoning	Which expectations and explorations can be grounded in branch-resident findings
IV. Automated-actor supervision	What an actor may do within the branch, and under what oversight the branch itself may be enacted

6.1 The branch

A branch is a graph state derived from another graph state by hypothetical events. It is not a new primitive. It uses the existing version and graph-state contract while declaring that its relationship to reality is subjunctive rather than observed.

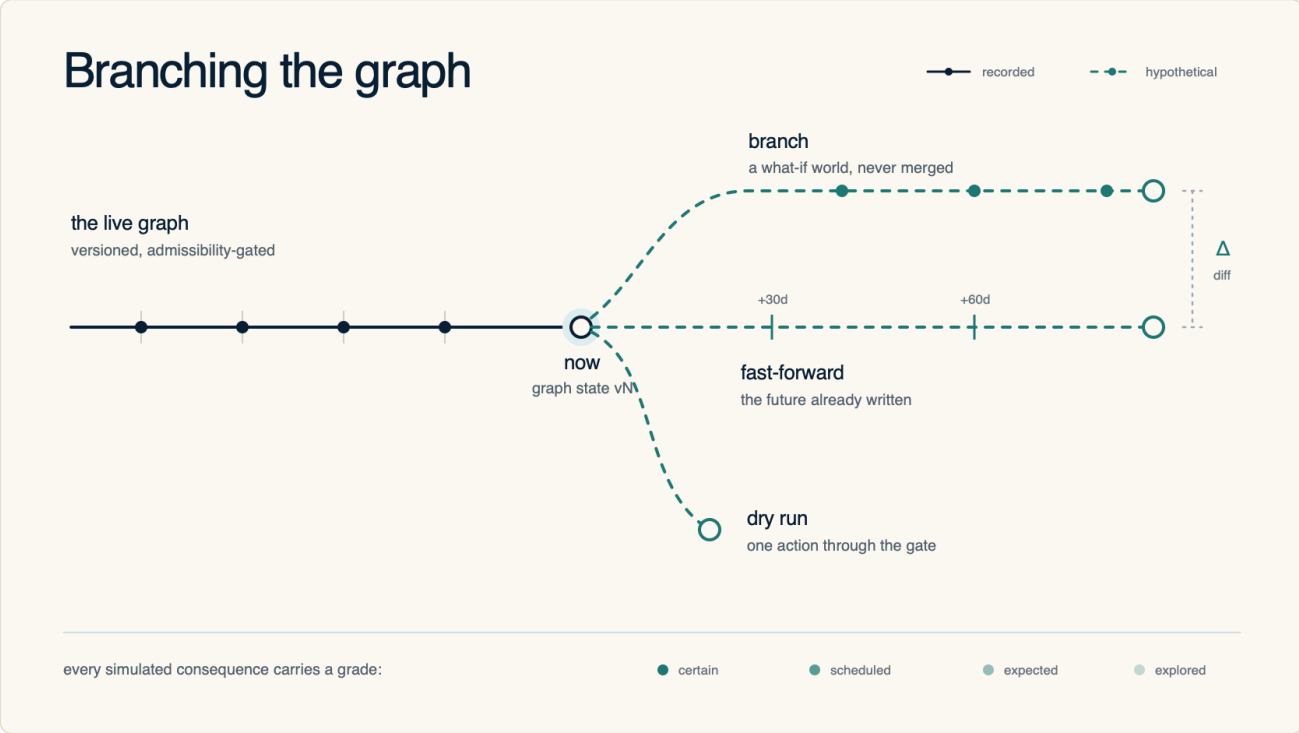


Figure 6. A branch evaluates hypothetical events against a versioned graph state without changing recorded reality.

A branch has three defining properties.

First, it forks from a specified graph state. The fork can be current, for a forward-looking decision, or historical, for a counterfactual review of what might have followed from a different past action.

Second, hypothetical events pass through the same conceptual evaluation pipeline as live proposals. They are typed, bound to citizens and actors, evaluated against declarations, authority and evidence, and propagated through the branch. Where the live pipeline would call an executor, the branch advances only its own hypothetical state.

Third, a branch never changes live enterprise facts. Its findings remain marked as hypothetical. Only a governed decision, plan, expected trajectory or diff returns to live operation.

6.2 Three branch modes

Mode	Source of future events	Question answered	Character
Dry run	One explicitly proposed action	What would this action do if proposed now?	Deterministic counterfactual evaluation
Scheduled fast-forward	Dated facts already resident in the graph	What becomes due, expired or inadmissible if nothing else changes?	Deterministic reading of the already-declared future

Mode	Source of future events	Question answered	Character
Generated branch	Behavioural envelopes, stochastic models, optimisation, human scenarios or simulated actors	What kind of enterprise state could this path create?	Exploratory simulation over authoritative structure

A dry run asks the control boundary for a verdict without requesting execution. An automated actor can use it to test a planned action, see which oversight mode it would receive, identify evidence gaps and inspect the obligations or consequences it would create.

A scheduled fast-forward advances time without inventing new behaviour. Certifications lapse, contracts renew, obligations fall due, deferrals expire and delegations end according to dated facts already present. This answers a common operational question "*what breaks in the next ninety days if nothing changes?*" without statistical prediction.

A generated branch introduces proposed future events from an explicit generator. The generator may draw on the behavioural envelopes citizens already carry, but it can also be external. DIG removes the need for that generator to reconstruct the enterprise's structural and normative model. The generator supplies possible events; the branch supplies the enterprise state, declarations and consequences against which they are resolved.

6.3 Evidence grades

Prediction is not a single epistemic category. DIG distinguishes what follows deductively from what is expected or explored.

Grade	Derived from	Character
Certain	Declarations and lifecycle conditions	Holds for every branch sharing the stated premises and evaluation version
Scheduled	Dated facts already in the graph	Occurs if the dated fact remains unchanged
Expected	Behavioural envelopes or declared statistical baselines	Estimate bounded by the citizen model and its assumptions
Explored	Generated or actor-driven paths	Conditional on generator, parameters and path taken

Every generated branch should identify its forked graph state, generator and version, assumptions, proposed event sequence, evaluation version and result grades. A KPI is then a query over the branched graph state and inherits the grade of the facts it aggregates.

The grade does not make an estimate correct. It prevents an estimate from being presented with the epistemic status of a declaration or dated fact.

6.4 The unscoped diff

A conventional simulation is usually built to answer a scoped question. A DIG branch evaluates the events supplied to it and then follows whatever typed dependencies those events reach. The resulting diff can therefore include findings outside the domain that motivated the branch.

A scheduling change may improve throughput while pushing a certification beyond its valid window, breaking a control's evidence cadence or placing a service commitment at risk. Those consequences appear only if the relationships have already been modelled; where they have not, the branch exposes a modelling gap rather than silently claiming completeness.

This is the value of the unscoped diff. It does not discover unknowable relationships. It refuses to stop at an arbitrary module boundary when known relationships cross it.

6.5 From branch to expectation

When a branch is adopted, its hypothetical events do not become recorded history. Instead, the selected branch or plan becomes an expected trajectory bound to the relevant live citizens. Arriving reality can then be evaluated against that expectation through hybrid conformance.

The resulting loop is:

1. Observe the live graph.
2. Branch a proposed change.
3. Read the graded diff and exposed modelling gaps.
4. Decide through the appropriate authority path.
5. Execute through the control boundary.
6. Monitor live citizens against the adopted expectation.
7. Branch again when reality diverges or new evidence arrives.

For an automated actor, rehearsal, governance and monitoring share the same actor identity, authority envelope, action ontology and evidence contract. The generator used in a branch may still differ from the live model or agent and must be versioned accordingly. What remains common is the enterprise substrate against which the rehearsal and live actions are judged.

6.6 What admissible futures do not claim

A branch is only as faithful as the graph state and generator from which it is produced. Certain and scheduled findings inherit the quality of their declarations and dated facts. Expected and explored findings remain estimates. Exogenous events not represented or generated do not appear. Resource, duration and stochastic behaviour require appropriate generators and validation.

DIG therefore does not claim to predict the future. It distinguishes which consequences follow from declared premises, which are already scheduled, which are expected under stated baselines, and which arise only in explored paths. Its contribution is that all four grades are evaluated against the same versioned enterprise structure used for live reasoning and control.

7. The Acme Wings scenario completed

The fictional Acme Wings example can now be followed through the full methodology.

7.1 Before DIG

Acme Wings operates under airworthiness requirements that specify who may perform maintenance, how long inspections can be deferred, which evidence must support return to service and how compromised evidence affects a decision.

The regulatory text, maintenance activity, inspection records, technician certifications and security events live in separate systems. When a security intrusion affects the maintenance tracking system, the security team initially owns the incident. For it to influence aircraft operations, people must recognise the dependency, identify the affected time window, find the relevant work orders, validate technician and inspection evidence, interpret the applicable requirements and notify the accountable maintenance authority.

The enterprise can perform this work, but its understanding is assembled after the event. During the assembly, affected decisions may remain apparently valid.

7.2 Representation in DIG

Under DIG, Acme Wings declares an authoritative scope covering maintenance work orders, aircraft release decisions, technician certification, evidence integrity and the execution surfaces that change airworthiness state.

The graph contains:

- Aircraft, components, technicians, work orders and source systems as citizens.
- Airworthiness clauses, role requirements, evidence conditions and deferral limits as declarations.
- Maintenance, inspection, signature, security and release events as imperative structure.
- Inspection records, system logs, signatures and attestations as evidence.
- Maintenance authorities and emergency roles as authority relations.
- Work orders and maintenance episodes as trajectories.

The scope declaration also identifies exclusions. If a supplier system remains outside DIG, a finding dependent on that system cannot be treated as complete.

7.3 Trigger and propagation

The intrusion is recorded as a state transition on the maintenance system citizen for a defined time window. Cross-domain reasoning follows the represented dependencies to the records created through that system, the work orders relying on them, the aircraft touched by those work orders and the declarations requiring trustworthy evidence.

Affected work orders receive findings such as *evidence insufficient* or *not evaluable* rather than automatically being labelled violated. The distinction depends on the declaration and

evidence available. Aircraft states and proposed release actions are then evaluated against those findings.

Each result retains a path to the source event, affected system state, record, work order, aircraft, declaration, evidence status and graph state version. The organisation can see both what is affected and why.

7.4 Governed response

An AI remediation assistant is represented as an actor. It may assemble affected work orders, request evidence and draft remediation plans. It may propose a reinspection or grounding event. It may not release an aircraft unless the required work-order state, technician certification, evidence integrity and authority conditions are satisfied.

If the assistant proposes direct closure because similar historical incidents were later resolved, the action ontology classifies the proposal as a closure action. The authority envelope does not permit it. The action is blocked or routed for approval according to the current oversight mode, and the attempt becomes a drift finding with evidence.

The important fact is not that the actor is an AI. A workflow bot or human user acting outside the same authority would be governed by the same structure.

7.5 Branching remediation

Before committing a remediation plan, Acme Wings can branch the current graph state. One branch reinspects every affected aircraft immediately. Another prioritises aircraft by dependency and scheduled utilisation. A future-event generator can supply duration and resource assumptions, while DIG adjudicates proposed events against certification, evidence and airworthiness declarations.

The branch diff reports operational KPIs and any cross-domain effects. Results derived from expiry dates are scheduled; resource projections are expected or explored. If a branch depends on missing supplier evidence, the gap appears in the result.

Once a plan is approved, it is executed through the governed operational systems and retained as an expected trajectory. Live work orders are evaluated against that expectation as the remediation proceeds.

The scenario illustrates the full DIG loop: represent, evaluate, propagate, govern, branch, execute and monitor.

8. Related approaches and the DIG contribution

DIG draws on established ideas. It does not claim to have invented graphs, declarative constraints, object centricity, state machines, event histories, digital twins, policy engines, simulation, provenance or graph-grounded AI.

Knowledge graph research provides models and languages for representing entities and relationships across domains [6]. Provenance standards represent the derivation of entities and

activities [7]. Event sourcing records state changes as event sequences that can reconstruct prior state [10]. Digital-twin frameworks define synchronised representations of operational entities and systems [11]. Access-control and zero-trust architectures govern requests using identity, attributes, resources and context [8, 9]. Process research brings declarative constraints, observed traces and object-centric structures into conformance analysis [1–3].

DIG's claimed contribution is the factoring of these ideas into one universal enterprise methodology: a shared primitive contract for declarations, live trajectories, stateful citizens, evidence and authority; a graph-native substrate in which those primitives are co-resident; four state-invariant reasoning classes operating over that substrate; a control boundary through which the fourth class governs consequential action; and a branch model through which all four determine admissible futures. To our knowledge, adjacent approaches can implement several of these properties, and individual products may combine more than their category normally implies. The distinction is the combination made primary by the methodology, not an assertion that other architectures are incapable of extension.

8.1 Architectural comparison

Approach	Declarative structure	Live events	Maintained object state	Cross-domain graph	Evidence path	Action boundary	Branched futures
Process mining and OCPM	Possible or process-scoped	Core	Partial or derived	Process-scoped	Varies	Outside primary scope	Possible through extensions
Enterprise or temporal knowledge graph	Core or possible	Possible	Possible	Core	Possible	Implementation-dependent	Possible
Rules engine or policy-as-code	Core	Inputs	Usually external	Implementation-dependent	Varies	Decision-focused	Usually external
Event sourcing or complex event processing	Usually external	Core	Derived or materialised	Often service-scoped	Event history core	Implementation-dependent	Possible
Digital twin or digital thread	Possible	Core	Core	Domain-dependent	Possible	Implementation-dependent	Common extension
Data or knowledge fabric	Semantic integration core	Possible	Source-dependent	Cross-domain	Varies	Outside primary scope	Outside primary scope
Simulation platform	Model-specific	Simulated	Simulated	Scope of model	Scenario provenance varies	No live enforcement by default	Core
DIG	Core	Core	Core	Core within declared scope	Core	Core for integrated actions	Core through graph state branching

The table describes category centres rather than hard limits. A knowledge graph can support rules, a digital twin can use a graph, a policy engine can record evidence, and a simulation platform can integrate live data. DIG does not depend on denying these possibilities. It pro-

poses that their shared semantic obligations should be designed once at enterprise graph level.

8.2 What DIG is not

DIG is not traditional process mining, though it includes an imperative layer that supports process mining questions. Process mining remains a discipline for discovering and analysing operational behaviour from event data.

DIG is not a workflow engine. It does not send the message, post the payment or move the work order. It evaluates whether a governed executor may do so.

DIG is not a rules engine. Rules engines can implement parts of declaration evaluation, but DIG also holds the citizens, trajectories, evidence, authority and graph state against which the rules are resolved.

DIG is not an ontology alone. Ontological structure is part of the graph, but DIG additionally maintains operational state, events, lifecycles, findings and action authority.

DIG is not a standalone simulation platform. It provides a branchable operational substrate. Future-event generators and simulation methods can operate against that substrate without reconstructing its structural model.

DIG is not a compliance certificate, legal conclusion, audit opinion or safety case. It makes the evidence and evaluation path behind such determinations explicit. Competent authorities remain responsible for the formal conclusion appropriate to the domain.

DIG is not primarily a model-alignment system. It does not make a language model deterministic or inherently safe. It grounds graph-derived claims and governs consequential actions through declared scope, authority, evidence and enforcement.

DIG is not a standalone product. Ahead implements the methodology on Neo4j. Implementations can begin alongside the existing estate and establish authority incrementally.

9. Adoption and governed growth

DIG adoption is not a single migration from many systems into one graph. It is the progressive establishment of authority over bounded domains and decision classes.

A useful starting scope has a cross-domain question that matters, a manageable set of citizens and declarations, identifiable domain owners, available evidence and a clear operational boundary. Acme Wings might begin only with evidence integrity for return-to-service decisions. A financial organisation might begin with one payment threshold and the authority chain around it. The methodology is universal; the first authoritative scope should be narrow enough to review and operate responsibly.

9.1 Establishing authority

A graph representation should not become authoritative merely because it has been extracted or generated. Authority is established through a governed lifecycle:

1. Identify the source document, system or accountable owner.
2. Map the relevant citizens, declarations, relationships and evidence requirements.
3. Review the representation with competent domain owners.
4. Record approval, scope, version and known exclusions.
5. Run the graph determination alongside the existing process where appropriate.
6. Reconcile differences and correct either the source or graph representation.
7. Activate authority for the declared scope.
8. Retain rollback, appeal and supersession paths.

This is especially important for declarations. Converting policy or regulatory prose into executable graph structure is an act of interpretation. DIG makes the interpretation explicit and governable; it does not eliminate the need for accountable judgment.

9.2 Growing through use

Once the scope is active, operational questions test its completeness. A stale source, unresolved identity, missing relationship, insufficient evidence type or undefined authority becomes a finding or limitation. Repeated gaps reveal where additional modelling would create value.

Not every gap should be filled. Some questions remain properly outside the graph's authority. Others may require data that cannot lawfully or practically be centralised. The growth loop therefore ends in a domain-owner decision: extend the scope, improve a source, represent an explicit exclusion, or retain the external determination.

This prevents "self-growing" from meaning uncontrolled semantic expansion. DIG helps grow itself by producing an evidence-based agenda for human-governed modelling work.

9.3 Maturity is multidimensional

An implementation can mature along several independent axes:

- **Coverage:** which citizens, declarations and trajectories are represented.
- **Freshness:** how current the connected sources and states are.
- **Evidence:** how completely findings are supported by recognised evidence.
- **Authority:** which decisions and actor relationships have been reviewed and activated.
- **Control:** which operational actions are actually gated before execution.
- **Replay:** which queries, graph states and evaluation versions can be reproduced.
- **Resilience:** which failure, override and recovery paths have been exercised.

A deployment can therefore be strong in modelling and weak in control integration, or authoritative for one domain while exploratory in another. Reporting these dimensions is more honest and useful than assigning DIG a single enterprise-wide maturity label.

10. Closing

The enterprise's divisions between regulations and operations, contracts and customers, controls and processes, people and authority are consequences of how software categories evolved. They are not properties of the enterprise itself.

DIG offers one modelling methodology across those boundaries. Declarative and imperative structure share a graph. Citizens bear state and governed lifecycles. Evidence, source, authority and version are part of the same primitive contract. The reasoning classes, automated action and branched futures operate against explicit graph states.

The methodology's strongest claims remain conditional. A graph cannot be more authoritative than the modelling, evidence, sources, governance and integrations that support it. The appropriate answer to missing coverage is an explicit limitation, not a fabricated verdict.

That limitation also creates the adoption mechanism. Each unresolved question can reveal a stale source, missing relationship, unevaluable declaration, absent evidence type or ungoverned execution surface. Once reviewed by competent owners, those findings identify where the model should grow. DIG becomes more complete through governed encounters with the boundaries of its present representation.

The organisational work is therefore inseparable from the architecture. A declaration becomes authoritative through domain-owner review, a versioned link to its source, evidence of approval, dual running where appropriate, and a rollback or appeal path. DIG makes that migration an explicit event rather than an assumption.

Initial implementations are now testing the methodology in bounded enterprise scopes. Their results will determine how its architectural claims behave under real constraints of data quality, modelling effort, latency, adoption and control integration. This paper defines the target against which that work can be assessed.

About Ahead ehf.

Ahead is an Icelandic enterprise technology company working at the intersection of process mining and graph-native knowledge infrastructure. DIG has emerged from that research and practice. Ahead has worked with leaders in process mining and knowledge graphs and is a partner to Neo4j.

About the author

Ragnar Láki Jónsson is co-founder and CTO of Ahead ehf. He leads Ahead's research and product direction, including DIG. Ragnar's prior work includes contribution to open-source declarative process mining [4].

References

[1] Dina Kretzschmann, Alessandro Berti, and Wil M. P. van der Aalst (2026). “State-Aware Object-Centric Process Mining: Enhancing OCEL 2.0 with Explicit State Transitions.” In *Enterprise Design, Operations, and Computing: 29th International Conference, EDOC 2025, Revised Selected Papers*, Lecture Notes in Computer Science 16213, pp. 103–118. Springer. https://doi.org/10.1007/978-3-032-15140-7_6.

[2] Tijs Slaats, Paul Cosma, and Thomas Hildebrandt (2026). “DisCoveR and Beyond: Exploring Recent Advances in Declarative Process Discovery.” In *Mining a Scientist’s Process: Essays Dedicated to Wil van der Aalst on the Occasion of His 60th Birthday*, Lecture Notes in Computer Science 16480, pp. 485–505. Springer. https://doi.org/10.1007/978-3-032-17618-9_33.

[3] Dennis M. M. Schunselaar, Tijs Slaats, Fabrizio M. Maggi, Hajo A. Reijers, and Wil M. P. van der Aalst (2018). “Mining Hybrid Business Process Models: A Quest for Better Precision.” In *Business Information Systems: 21st International Conference, BIS 2018*, Lecture Notes in Business Information Processing 320, pp. 190–205. Springer. https://doi.org/10.1007/978-3-319-93931-5_14.

[4] Simon V. H. Hermansen, Ragnar Jónsson, Jonas L. Kjeldsen, Tijs Slaats, Vlad Paul Cosma, and Hugo A. López (2024). “DCR4Py: A PM4Py Library Extension for Declarative Process Mining in Python.” In *Proceedings of the ICPM 2024 Tool Demonstration Track*, CEUR Workshop Proceedings, Vol. 3783. https://ceur-ws.org/Vol-3783/paper_353.pdf.

[5] Luca Nannini, Adam Leon Smith, Michele Joshua Maggini, Enrico Panai, Sandra Feliciano, Aleksandr Tiulkanov, Elena Maran, James Gealy, and Piercosma Bisconti (2026). “AI Agents Under EU Law: A Compliance Architecture for AI Providers.” Working paper, arXiv:2604.04604v1. https://arxiv.org/abs/2604.04604.

[6] Aidan Hogan et al. (2021). “Knowledge Graphs.” *ACM Computing Surveys*, 54(4), Article 71. https://doi.org/10.1145/3447772.

[7] Timothy Lebo, Satya Sahoo, and Deborah McGuinness, eds. (2013). “PROV-O: The PROV Ontology.” W3C Recommendation, 30 April 2013. https://www.w3.org/TR/prov-o/.

[8] Vincent C. Hu, David Ferraiolo, D. Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone (2014; updated 2019). *Guide to Attribute Based Access Control (ABAC) Definition and Considerations*. NIST Special Publication 800-162. https://doi.org/10.6028/NIST.SP.800-162.

[9] Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly (2020). *Zero Trust Architecture*. NIST Special Publication 800-207. https://doi.org/10.6028/NIST.SP.800-207.

[10] Martin Fowler (2005). “Event Sourcing.” https://martinfowler.com/eaaDev/EventSourcing.html.

[11] International Organization for Standardization (2021). *ISO 23247-1:2021, Automation systems and integration—Digital twin framework for manufacturing—Part 1: Overview and general principles*. https://www.iso.org/standard/75066.html.

[12] Ian Robinson, Jim Webber, and Emil Eifrem (2015). *Graph Databases: New Opportunities for Connected Data*, 2nd edition. O’Reilly Media. https://www.oreilly.com/library/view/graph-databases-2nd/9781491930885/.

[13] International Organization for Standardization and International Electrotechnical Commission (2024). *ISO/IEC 39075:2024, Information technology: Database languages: GQL*. https://www.iso.org/standard/76120.html.

Citing this paper

Ragnar Láki Jónsson (2026). *DIG: The Declarative-Imperative Graph: A universal modelling methodology for the enterprise in the AI era*. Ahead ehf., November 2026.

© 2026 Ahead ehf. This document describes a methodology. Proprietary formal semantics, implementation details, internal architectures and client-specific models are not disclosed.
Cover image: Timothy P. Jenkins, DTU