The Violation Situation Pattern: Persistent Representation of Compliance Violations in Knowledge Graphs

Nima Kamali Lassem[1,*], Fuqi Song[1] and Seyid Amjad Ali[2]

[1]DiliTrust, Paris, France
[2]Department of Information Systems and Technologies, Bilkent University, Türkiye

Abstract

In query-based compliance checking over knowledge graphs, detection results alone do not provide persistent violation identities, review states, or audit histories. We address this representation gap with the Violation Situation Pattern (VSP), a reusable ontology pattern that extends the Situation pattern of Gangemi and Mika by modeling each detected violation as a persistent first-class graph entity. Each violation is associated with a rule identifier, temporal validity interval, lifecycle state, and evidence links to the affected entities, while immutable lifecycle events provide a queryable history of lifecycle transitions. We instantiate VSP on the schema of a deployed legal knowledge graph integrating corporate governance and contract data, populated from public contract corpora with planted violations providing ground truth, and operationalize four deontic compliance rules. We verify implementation stability under repeated rule execution and scope expansion, observing no duplicate violations and preservation of existing identities and transition histories in the tested scenarios. An exploratory analysis of 73 regulatory enforcement decisions separately illustrates increased rule-body scope coverage. These results support continuity under the tested revisions; they do not establish detection accuracy or suitability for operational investigations. The complete implementation, queries, SHACL shapes, and evaluation dataset are publicly available to support the reproducibility of the reported results.

Keywords

ontology design patterns, knowledge graph, legal compliance, violation lifecycle, provenance, audit, deontic logic, SHACL

1. Introduction

This paper addresses the representation of findings produced by rule- and query-based compliance checks over knowledge graphs. When findings are returned without persistent violation records, repeated executions re-derive matches without retaining their review status or resolution history. External ticketing or case-management systems can preserve this information, but answering audit questions then requires linking those records to the domain entities in the graph. We investigate how violation identity, lifecycle, and history can instead be maintained within the knowledge graph itself.

This limitation becomes apparent in a simple compliance scenario. A signature-authority breach is detected for a contract in January and subsequently reviewed and dismissed. When the rule is executed again in April, the same breach is detected once more. An auditor later asks when the issue was first identified, who dismissed it, and whether it has recurred since. Detection outputs alone cannot answer these questions because the January detection, the review decision, and the April re-detection were never captured as the lifecycle of a single persistent violation.

Throughout this paper, compliance rules are expressed in a deontic contract logic, the knowledge graph is implemented as a property graph, and portability is assessed using the Shapes Constraint Language (SHACL). Existing compliance approaches, including SHACL-based validation [1] and deontic reasoning over reified General Data Protection Regulation (GDPR) representations [2], effectively detect violations when the required information is available. Our focus is the persistent identity and review history of the resulting findings, beyond the determination that a rule has been violated. Knowledge

The 25th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2026) – Satellite Events (Workshops, Tutorials, Demos, Posters).

*Corresponding author.
nima.kamali@dilitrust.com (N. Kamali Lassem); fuqi.song@dilitrust.com (F. Song); syedali@bilkent.edu.tr (S. A. Ali)
0009-0008-4326-9639 (N. Kamali Lassem); 0009-0001-4900-5369 (F. Song); 0000-0001-9250-9020 (S. A. Ali)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-26

CEUR-WS.org/Vol-4340/paper16.pdf

graphs represent domain concepts as persistent semantic objects [3], and legal knowledge graphs extend this principle to contracts, clauses, persons, and signatures [4, 5]. Where findings are not retained as persistent graph entities, a rule match alone does not record review decisions or connect repeated detections to a common violation history.

To address this limitation, we introduce the Violation Situation Pattern (VSP), which represents every detected violation as a persistent graph node. Each violation is associated with a rule identifier, a temporal validity interval governed by a staleness constraint, a finite-state lifecycle for review and remediation, and evidence relations linking the violation to the relevant contracts and persons. VSP builds on the Situation Ontology Design Pattern (ODP) proposed by Aldo Gangemi and Peter Mika [6] while extending it with lifecycle management, staleness handling, and explicit identity conditions required for compliance auditing. Every lifecycle transition is recorded as an immutable graph event, allowing the recorded lifecycle history of a violation to be reconstructed through graph traversal.

Representing violations as persistent graph entities directly expands the questions that a compliance knowledge graph can answer. Repeated detections are linked to an existing violation rather than creating duplicate records, review decisions become part of the graph together with the responsible actor, rationale, and timestamp, and audit queries can be answered directly within the same graph that stores contracts, persons, and other domain entities. Building on these capabilities, this paper makes the following contributions:

C1. We introduce the Violation Situation Pattern (VSP), a reusable ontology pattern that represents each compliance violation as a persistent graph node with a stable identity, temporal validity interval, lifecycle state, and evidence relations to the entities referenced by the governing rule.

C2. We define a lifecycle model in which every state transition is recorded as an immutable event containing the responsible actor, rationale, and timestamp, enabling complete review histories to be reconstructed through graph traversal.

C3. We verify the implementation's stability under repeated rule execution and scope expansion: existing violations retain their identities and transition histories, while newly in-scope findings receive separate records.

C4. We analyze the scope coverage of two V4 rule bodies against 73 regulatory enforcement decisions, illustrating how a broader detector can reach additional findings without changing the pattern.

C5. We verify cross-formalism agreement for the V3 and V4 detection conditions using SHACL; persistent identity and lifecycle enforcement are outside this comparison.

Together, these contributions support auditing of persistent violation records as compliance rules evolve over time. The remainder of the paper is organized as follows. Section 2 defines the problem in legal compliance systems and identifies the representation requirements considered here. Section 3 positions VSP within compliance management, norm formalization, situation modeling, and provenance. Section 4 introduces the Violation Situation Pattern, Section 5 operationalizes four compliance rules, Section 6 reports stability, scope-coverage, and detection-parity checks, Section 7 discusses limitations, threats to validity, and future work, and Section 8 concludes the paper.

2. Background and Problem Setting

Compliance is managed as an organizational process spanning strategy, processes, technology, and people [7], and research on business process compliance has concentrated on aligning regulatory rules with the design, verification, and validation of processes [8]. Methods include model checking, declarative constraints such as Declare, and process mining [8]. VSP concerns the resulting findings; upstream rule extraction, whether through natural language processing (NLP) or large language models (LLMs), is outside its scope. In operational legal-tech systems this takes the form of an alert-driven workflow: rules run as queries over contracts, governance data, or process records, and the resulting

alerts pass to compliance personnel for review, escalation, dismissal, and remediation. Compliance practice separates these functions at the tool level. The compliance management ontology of Abdullah et al. [9], built from practitioner interviews and eight organizational case studies with heads of compliance and compliance managers, models review and record keeping as program activities of their own and distinguishes review tools and record-keeping and reporting tools from obligations-management tools. In the deployments this work draws on, the consequence is that a finding's review state is held in ticketing or case-management systems while the contracts, mandates, and persons it concerns are held in the graph. During regulatory review, however, an organization must demonstrate when a violation was first detected, who assessed it, and how it was resolved, and that history is scattered across emails, spreadsheets, and case-management tools.

For the audit questions considered here, we identify three representation requirements: stable identity, lifecycle management, and immutable provenance. These reflect operational requirements communicated by DiliTrust's legal and product teams, detailed through the competency questions in Section 4.1; they are not an exhaustive specification of a compliance system. Abdullah et al.'s review and record-keeping constructs [9] motivate tracking review state and decision history; stable identity connects these across repeated detections. **Stable identity** ensures that re-evaluation recognizes an existing violation rather than creating a duplicate. **Lifecycle management** records the current review stage. **Immutable provenance** makes every decision part of a reconstructible audit history. When these records are maintained separately from the domain graph, audit questions require correlating them with the entities the rule concerns and maintaining those links as rules change. VSP brings these records together: the violation record lives in the same graph as the contracts, clauses, and persons it concerns.

DiliTrust's Legal Entity Management (LEM)[1] captures corporate governance facts: companies, governance bodies, mandates with appointment and cessation dates, and signing authorities. Contract Lifecycle Management (CLM)[2] captures contract-level facts, including contract metadata, contracting parties, signatures, and clauses by typology. Clause-boundary and typology extraction that feeds CLM ingestion is inspired by contract NLP benchmarks such as CUAD [10] and ContractNLI [11]. Graphs of this shape support entity-level queries (who serves on which board, which contracts a company has signed), but compliance checks are queries over those entities rather than facts within the graph. A query that returns a contract signed without an active mandate surfaces a fact; that fact does not enter the data as a persistent object.

3. Related Work

Racz et al. [7] and Hashmi et al. [8] frame compliance management and its operational tasks. Abdullah et al. [9] model corrective controls for detected non-compliance. These works motivate VSP's focus on the finding itself: its identity across repeated detections, review state, and links to evidence.

A second line of work formalizes the norms themselves so that non-compliance can be derived. Formal Contract Logic (FCL), introduced by Governatori and Milosevic [12], represents contractual obligations in a defeasible deontic setting and accounts for the secondary obligations that arise once a primary obligation is breached; the same formalism has been applied to checking business processes against contract constraints [13]. We adopt FCL for the rule layer of this work for that reason: it expresses the obligation structure of our rules directly, and it keeps the rule body separable from whatever stores the result. Robaldo et al. [2] formalize GDPR provisions in reified input/output logic and publish them as the DAPRECO knowledge base, focusing on the formal representation of norms. SHACL-driven contract compliance checking [1] produces validation reports. Such reports can be retained; persistence alone, however, does not specify how repeated findings share an identity or accumulate a review lifecycle. VSP addresses these representation choices independently of the detection formalism.

Within ontology engineering, the reification of contexts is well established. Gangemi and Mika [6]

[1] https://www.dilitrust.com/legal-entity-management-roi-calculator/
[2] https://www.dilitrust.com/dilitrust-governance-suite/contract-management/

introduce Descriptions and Situations, a two-level architecture in which a Description is a non-physical entity such as a norm or a plan, and a Situation is a reified state of affairs that satisfies it through structured constituents. VSP inherits this architecture directly. The deontic rule plays the role of the Description, and each detected violation is the Situation that satisfies it. We do not reify the Description inside the graph: keeping the rule external, referenced only by identifier, is what allows a rule body to be revised without disturbing the violations already recorded under it. Carriero et al. [14] model recurrent situations as a series with unifying factors. Repeated detections could likewise be represented as separate events. VSP instead groups them under one finding identity and records lifecycle changes in its transition log. The distinction is the unit of representation: a detection event versus the finding that persists across detections.

Closer to the audit motivation of this work, Waltersdorfer and Sabou [15] argue that knowledge graphs can address the fragmentation of audit evidence in AI system governance, and they set out an architecture and prototypes for integrating audit data semantically. VSP addresses the more specific question of identifying repeated violation findings and recording their review lifecycle. At the infrastructure level, Dibowski [16] presents PROV-STAR, a Resource Description Framework (RDF)-star extension of PROV-O that intercepts updates and records every inserted and deleted triple with its timestamp and agent, so that any prior state of a graph can be recovered. PROV-STAR tracks mutations of triples in a domain-agnostic way; VSP records domain events, where each entry in the log is a named lifecycle transition of a violation rather than a raw edit. The two are compatible, and a PROV-STAR style substrate would be a reasonable persistence layer for the transition log described in Section 4.

Knowledge graphs have also been applied to the norms themselves. Hernandez et al. [17] introduce the TAIR ontology, which extracts normative statements from the EU AI Act and from harmonized ISO standards and publishes them as linked Requirement and Concept instances that can be queried for cross-document consistency. TAIR operates on regulatory text and reports no enforcement instances; whether a given organization has violated a requirement is outside its scope. The relation to VSP is one of layering rather than competition. A requirement represented in a norm graph of this kind is a natural source for the rule identifier that a violation node carries, and the two representations can be linked without either absorbing the other. A parallel effort from the ontology-patterns workshop WOP 2025, ARGOS [18], targets governance patterns for LLM-driven applications and data operations; it governs the operations that produce findings rather than the findings themselves.

One further body of work sits outside the research literature. Event sourcing — recording state as an append-only sequence of events and deriving the current state by replaying them — is established engineering practice, and VSP's transition log is structurally an event log of that kind. The resemblance is real, but the problems differ. In event sourcing an object's identity is fixed by the command that creates it, so the log stores an asserted fact, and revising the schema of those events is correspondingly expensive, requiring conversion or replay of the store [19]. A violation is not asserted but derived: it exists because a rule matched, and the rule is revisable. VSP therefore keys identity on the referents a rule concerns rather than on the rule that produced the match — which is what lets a rule body be rewritten while the findings recorded under it keep their identity and history, with no conversion and no replay.

Read together, these lines of work divide along a consistent boundary. The norm-layer approaches represent obligations, the detection approaches decide whether an obligation has been breached, the governance literature prescribes what an organization then does, the provenance approaches record how a graph changed, and engineering practice preserves asserted state. What none of the approaches surveyed here provides is a representation of the finding that sits between them: an object with an identity stable across repeated evaluations of the same rule, a lifecycle recording the review and remediation decisions taken about it, an append-only history of those decisions, and evidence relations to the entities the rule concerns. In an operational legal graph it is this object that an auditor asks about, and this object that is absent. VSP supplies it at the application level, and it does so without prescribing how detection is implemented or where the graph is stored.

Before defining the pattern we make the reuse strategy explicit. Several catalogue patterns supply part of what a violation record requires, and Table 1 summarizes which part each one supplies. The

comparison is offered as design rationale rather than as evidence of a gap in the catalogue: it records why no single pattern could be adopted unchanged, and which constituents of VSP are inherited rather than new. These criteria describe capabilities, not individual components of the tuple in Section 4.1.

Table 1
Reuse analysis of catalogue patterns. Columns: ID = stable identity across re-evaluations; LC = finite-state lifecycle with named transitions; AT = immutable audit/transition log; EV = multi-referent evidence cardinality. ✓ = directly provided; ∼ = partially provided; ✗ = not provided.

Pattern (source)	ID	LC	AT	EV	Constituent supplied / not supplied
Situation / Description+Situation [6]	∼	✗	✗	✓	No identity, lifecycle, or transition log
EventCore [20]	✗	✗	✗	∼	Point-in-time only; no lifecycle or AT log
RecurrentSituationSeries [14]	✗	✗	∼	∼	Series-level recurrence structure; no per-instance identity or lifecycle
TimeIndexedSituation [21]	∼	✗	✗	✓	Adds T; no S or AT log
TimeIndexedPersonRole [21]	✓	✗	✗	∼	Identity per role-period; no lifecycle
N-ary Relations [22]	✓	✗	✗	✓	Closest structural sibling; no LC or AT log
PROV-O `prov:Activity` [23]	∼	∼	✓	∼	Provenance trail; no native LC states or transition table per situation
VSP (this paper)	✓	✓	✓	✓	Composite identity key, five-state automaton, append-only transition log, rule-specific evidence edges

4. The Violation Situation Pattern

4.1. Formal Definition

Detection creates the violation record; review and remediation transition it; auto-revalidation re-confirms it or, on re-detection after closure, reopens it. We refer to the reified object as a violation record, and to the class that realizes it in a property graph as ViolationInstance.

We define the Violation Situation Pattern as the tuple

$$\mathrm{VSP} = (r, T, S, E) \tag{1}$$

where r is the FCL (Section 3) rule identifier the violation instantiates, $T = [\texttt{detectedAt}, \texttt{validUntil}]$ is the temporal validity interval, $S \in \{\text{DETECTED, CONFIRMED, UNDER_REVIEW, REMEDIATED, DISMISSED}\}$ is the lifecycle state, and E is a set of evidence edges binding the violation to its referent entities.

Following the ODP catalogue style of Presutti and Gangemi [21]:

Context. A knowledge graph in which a recurring rule-driven check produces detection events that consumers need to query historically without re-executing the rule. Nothing in that description is specific to legal compliance; it is what a domain must satisfy to instantiate the pattern.

Solution. Represent each distinct violation as a ViolationInstance node carrying (r, T, S, E). Pair it with an immutable LifecycleTransition class so the projection (current ViolationInstance state) and the event log (LifecycleTransition history) are both graph objects. Enforce two invariants at the application layer: only transitions in the lifecycle automaton are accepted, and `detectedAt` is immutable.

Consequences. The realization carries two tradeoffs. Both invariants are enforced at the application layer rather than the schema level, so a writer that bypasses the mediator of Section 4.2 can violate either. And a property-graph realization does not provide Web Ontology Language (OWL) entailment, so class and property inferences available in an RDF stack are not available here. Section 6.3 verifies agreement between property-graph and SHACL detection results; it does not test RDF-based lifecycle enforcement.

Competency questions. The composition of (r, T, S, E) supports audit questions that stateless detection cannot answer in a single traversal. The questions below were derived from requirements communicated by DiliTrust's legal and product teams, and confirmed by them as representative of what is asked in practice; they correspond to the accountability demands set out in Section 2, where an organization must show when a violation was detected, who assessed it, and how it was resolved. Four are representative:

CQ1. Was a given violation previously dismissed and later re-detected? A path query over the transition log for a DISMISSED-to-DETECTED sequence.

CQ2. Which violations were active at a given time t? A window query over violations with $\texttt{detectedAt} \leq t \leq \texttt{validUntil}$.

CQ3. Which entities are jointly implicated in two or more open violations? A multi-hop join across evidence edges with a threshold on the count.

CQ4. What is the complete review history of a given violation? An ordered retrieval of its transition log, returning actor, timestamp, and reason.

Section 6.1 reports their execution against the evaluation graph.

Properties under re-evaluation. Two properties follow from the identity key and the append-only log. First, re-evaluation is idempotent: because a violation is written through an upsert on its composite identity key (Section 4.2), a repeated detection of the same violation matches the existing node and extends its validity interval rather than creating a second node. Second, recorded lifecycle history is reconstructible by traversing the append-only transition log in timestamp order (Section 4.2). No single catalogue pattern in Table 1 supplies all four of stable identity across re-evaluations, a finite-state lifecycle, an immutable transition log, and multi-referent evidence. Specifying them together on one situation node, with the rule left external and therefore revisable, is the pattern's contribution.

4.2. Graph Realization

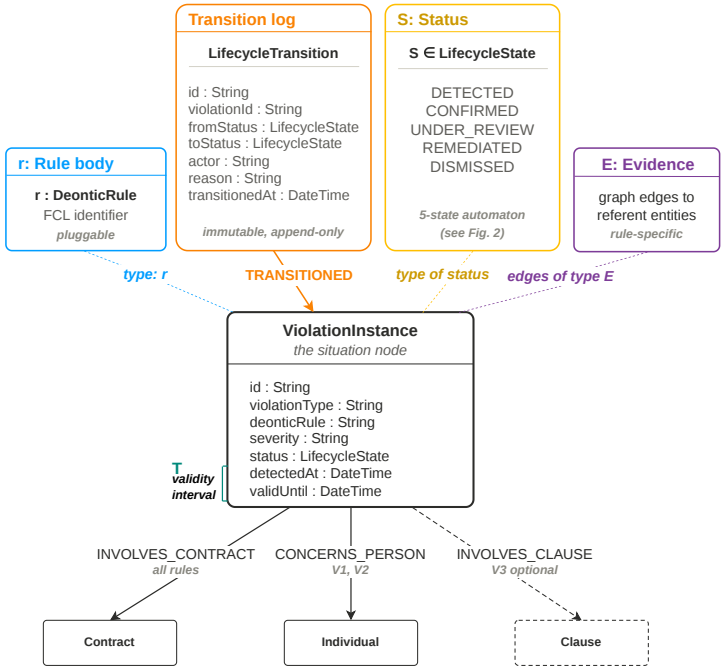


Figure 1: Pattern components (r, T, S, E) around the ViolationInstance node. Evidence cardinality is rule-specific; per-rule details are in Section 5.2.

The tuple defines the pattern abstractly; the property-graph realization below is one instantiation. As Fig. 1 shows, VSP realizes in a property graph as two node classes — ViolationInstance, the situation node, and LifecycleTransition, the immutable event-log node — linked by a TRANSITIONED edge, together with domain-specific evidence edges (in the legal domain, INVOLVES_CONTRACT, CONCERNS_PERSON, INVOLVES_CLAUSE).

The node carries `deonticRule`, the identifier r of the rule that first produced the record; subsequent matches retain this reference. It also carries a `violationType` category and a `severity` label. Identity is composite: $(\text{violationType}, \text{contract_id}, \text{person_id})$ for V1/V2 and $(\text{violationType}, \text{contract_id})$ for V3/V4 — keyed on the category and its referents, not on `deonticRule`, which is what leaves the rule body free to change. Detection writes each violation through an upsert keyed on this identity: a write that creates the node if no node carries that key and updates the existing one otherwise. On the first match the node is created with its detection timestamp; on any later match the same node is retained, its validity interval is extended, and its detection timestamp is never modified. Repeated matches preserve identity but do not create separate observation records. Re-detections that leave the lifecycle state unchanged therefore cannot be individually reconstructed; re-openings recorded as lifecycle transitions remain queryable.

Two invariants govern every subsequent write. Only arcs of the lifecycle automaton (Section 4.3) are committed, and `detectedAt` is immutable across an instance's lifetime. All state changes pass through a single write path, the `transition()` mediator, which accepts a change only if it is a permitted arc and never alters `detectedAt`. Because a property graph does not enforce these conditions at the schema level, the mediator enforces them in the application layer, and a caller bypassing it with a direct write can violate either invariant or both; Section 7.2 discusses stronger enforcement using SHACL and RDF [24, 25] as future work.

4.3. Lifecycle Automaton

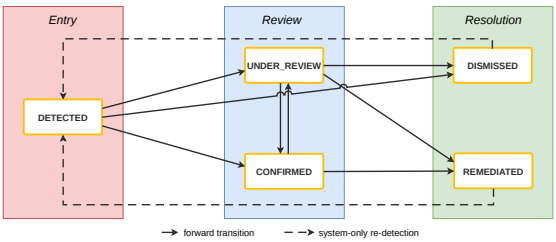


Figure 2: The five-state lifecycle automaton. Dashed arrows are the two system-only re-detection arcs, from REMEDIATED and DISMISSED back to DETECTED.

The state machine has five states and ten arcs, shown in Fig. 2: from DETECTED to UNDER_REVIEW, CONFIRMED, or DISMISSED; from UNDER_REVIEW to CONFIRMED, REMEDIATED, or DISMISSED; from CONFIRMED to UNDER_REVIEW or REMEDIATED; and from DISMISSED or REMEDIATED back to DETECTED. Actor type is recorded on each LifecycleTransition node rather than constrained per arc: arcs into UNDER_REVIEW and DISMISSED are taken by a human reviewer, the rest by either reviewer or revalidator. Each accepted transition writes a LifecycleTransition node with a TRANSITIONED edge to the ViolationInstance. The revalidation cycle re-executes each rule on a schedule. Staleness handling lets the validity interval lapse without renewed detection; if a closed violation's condition holds again, the re-detection arc reopens it.

Re-detection with the same composite key reopens the existing violation as a new review episode, preserving the earlier dismissal, reviewer, and rationale in the transition log. The direct arc from UNDER_REVIEW to REMEDIATED allows a correction to be recorded before review concludes, without requiring prior confirmation.

5. Rule Operationalization

Each rule is translated into a graph query and written through the identity-preserving upsert of Section 4.2, which instantiates a persistent violation record. V1 and V2 are expressed as FCL obligations, V3 and V4 through the clause-presence operationalization below. Table 2 lists the rules. The four were selected to exercise different parts of the pattern rather than to cover a legal area exhaustively, varying along three axes:

Subgraph and evidence cardinality. V1 and V2 operate on the governance subgraph and attach two evidence referents, a contract and an individual; V3 and V4 operate on the contract subgraph, V3 attaching an optional clause referent and V4 a single contract referent.

Normative source. Legally motivated operational controls for V1/V2; a policy control for V3; a statutory clause-presence proxy for V4.

Rule form. The full FCL expression for V1 and V2; the clause-presence operationalization for V3 and V4, used because the data layer represents clauses by `clauseType` rather than by structured obligation flags. We discuss this rule-correctness boundary in Section 5.2.

V4 carries the external evaluation because it is the only rule for which regulator-labeled ground truth is publicly available.

Table 2
Four implemented rules and their normative context. V1/V2 are operational authority checks motivated by company law, V3 is a policy control, and V4 operationalizes statutory requirements through clause presence (Section 5.2). Pattern semantics do not depend on normative source.

Rule	Type	Source
V1 Unauthorized Signature	operational	Company powers: Dir. (EU) 2017/1132 Art. 9; CdC L225-35
V2 Expired Signatory Mandate	operational	Recorded mandate expiry; cf. CdC L225-18 (directors' terms)
V3 Missing Confidentiality Clause	policy	Contractual control (commercial practice)
V4 Missing DPA Breach-Notification	statutory	GDPR 28(3)(f), 33; SCC 2021/915 [26] Annex, Clause 9

CdC: French Code de Commerce. DPA: Data Processing Agreement. SCC: Standard Contractual Clauses.

5.1. V1 and V2: Signature-Authority Violations

V1 (unauthorized signature) checks that every signatory holds a mandate at the contracting party whose scope includes SIGNATURE_AUTHORITY and whose temporal validity covers the signing date; a violation is the negation of that consequent for any non-OTHER signatory role. Detection identifies signatures by individuals whose AUTHORIZED_BY mandates either lack the required scope or are out of temporal validity at signing. V2 (expired signatory mandate) differs in that the signer once held the required mandate: detection finds an AUTHORIZED_BY mandate whose end date precedes the signing date and confirms no replacement mandate covers it. The write step is structurally identical, differing only in `violationType`. These are operational checks of recorded authority, not determinations of contract enforceability. Company-law provisions distinguish internal powers from effects on third parties, and directors' terms do not govern every signatory role. The two rules distinguish missing authority from recorded expiry.

5.2. V3 (Missing Confidentiality) and V4 (Missing DPA Breach-Notification)

V3 flags contracts lacking a CONFIDENTIALITY clause. The evaluation set comprises commercial contracts and contains no non-disclosure or data-processing agreements, for which the check is not meaningful. V3 is policy-derived: confidentiality is a standard contractual control rather than a statutory obligation

and is labeled a violation for uniformity with V1/V2/V4 rather than normative parity. V3 demonstrates that VSP applies identically to policy-derived rule bodies: the pattern's identity, lifecycle, and evidence semantics are independent of the rule's legal status. V4 flags DPAs lacking a BREACH_NOTIFICATION clause per SCC 2021/915 Annex, Clause 9. V4 detects clause presence rather than clause compliance. A DPA with a 90-day notification window passes V4 but violates GDPR Art. 33's 72-hour requirement. This is a rule-correctness boundary, not a pattern boundary; Section 6.2 quantifies the share of regulator-issued findings this operationalization covers.

Evidence cardinality is rule-specific: V1 and V2 attach INVOLVES_CONTRACT and CONCERNS_PERSON edges to CONTRACT and INDIVIDUAL nodes respectively; V3 attaches INVOLVES_CONTRACT and, when the contract carries clause nodes, an INVOLVES_CLAUSE edge to a CLAUSE node; V4 attaches INVOLVES_CONTRACT only.

6. Evaluation

The graph, the detection queries, the SHACL shapes, the evaluation datasets, and the scripts that produce every result in this section are available as a public artifact.[3] Each result below is reproduced by running the corresponding script against the released graph.

The evaluation supports the pattern claim through three pieces of evidence of distinct types: a direct check of the pattern's stability properties, a rule-body scope coverage analysis for V4 against GDPRhub decisions, and a cross-formalism portability check on V3/V4.

The evaluation graph instantiates the LEM/CLM schema of Section 2 but is populated with contracts from public corpora — CUAD [10], ContractNLI [11], and the EU Standard Contractual Clauses [26] — rather than with client data. Those corpora carry no corporate-governance records, so the companies, individuals, mandates, and signing authorities are generated, and violations are planted so that ground truth is known for every case. It carries 291 COMPANY, 269 INDIVIDUAL, 198 MANDATE, 235 CONTRACT, 331 CONTRACTPARTY, 100 CLAUSE, and 140 VIOLATIONINSTANCE nodes, plus 19 LIFECYCLETRANSITION nodes from a deterministic lifecycle exercise reaching all five states. V1, V2, and V3 are scoped through a manifest that partitions contracts into an authorized group, a planted-violation group, and an expired-mandate group; the partition is what makes ground truth available, so the detection counts below are counts over that partition. The stability checks of Section 6.1 do not depend on it, and are reported unscoped where the query planner permits.

6.1. Pattern Stability

The pattern's central claims are that re-evaluation does not duplicate a violation, that the audit log survives a change to the rule body, and that a violation keeps its identity and history when a rule's scope is widened.

The first two follow from the construction: an upsert on a composite key cannot create a second node for a key that already exists, and a log that is only ever appended to cannot lose an entry. What is not guaranteed is that the implementation realizes them, and that they survive operations which could plausibly break them. Each check below runs against the canonical graph, restores it on completion, and is reproducible from the artifact. Idempotence is measured by re-running every detection rule and comparing the violation set before and after; log immutability by hashing every transition node across both the re-evaluation and the rule revision.

Re-evaluation left the graph at 140 violations with no node created or lost and no `detectedAt` altered; all 19 transition-log entries survived both the re-evaluation and the rule revision unchanged. Because idempotence is a property of the write path rather than of detection correctness, it does not depend on the manifest partition, and re-running it unscoped over the full contract set — 342 violations rather than 140 — likewise created and lost no node and altered no `detectedAt`. Separately, the

[3] https://doi.org/10.5281/zenodo.21769799

mediator was exercised on twenty transition attempts, ten permitted arcs and ten violating ones; all ten permitted arcs committed and all ten violating attempts were rejected.

Co-location. The four competency questions of Section 4.1 execute as single queries against the evaluation graph and are released with it. This is the co-location claim in operational form: Waltersdorfer and Sabou identify the fragmentation of audit evidence across systems as the obstacle to acting on governance obligations [15], and none of the four is answerable from detection outputs alone. CQ1 and CQ4 ask for a review history that stateless detection never records, CQ3 needs the lifecycle status that separates an open finding from a closed one, and CQ2 asks about a past instant that re-executing the rule cannot reconstruct. One caveat: CQ3's multi-entity join returns results only for contracts, because each signatory in the evaluation set carries exactly one violation, leaving the person arm nothing to match.

The third claim is the substantive one, because it could have failed. Keeping the rule external and referenced only by identifier is a design decision, and whether violations survive a revision of the rule they were recorded under is a consequence of that decision rather than of the upsert. The rule revision widened V3's scope from its manifest to the full contract taxonomy: the 50 original violations retained identity and history, 135 new violations appeared only for newly in-scope contracts, and re-running the original rule produced no duplicates.

6.2. Rule-Body Scope Coverage: V4 on GDPRhub Decisions

Scope. Section 6.1 verifies identity and history preservation under V3 scope expansion. Here, V4 illustrates a separate question: how much of the enforcement evidence two rule bodies encompass. This analysis does not execute V4 lifecycle transitions or independently test audit continuity.

The revision itself widens what the rule body reaches. On 73 GDPRhub Article 33 enforcement decisions, extending V4 from clause presence (Rule A) to clause presence plus late notification (Rule B) raises the share of regulator-confirmed findings the body encompasses from 0.185 to 0.431 (Fig. 3). Both shares rest on subtype labels assigned by practitioners independent of the rule-body design, but without a second annotator or an agreement measure (Section 7). Both bodies are evaluated as predicates over evidence fields extracted from the decisions — whether a notification was made, and how long it took — so Rule B's deadline arm is a comparison against a threshold rather than a lookup of a subtype label. Coverage is unchanged at thresholds of 24, 48, and 72 hours, so the figure does not depend on the threshold chosen. This remains a scope-coverage measurement rather than an executed detection run: the decisions are regulator case records, not contracts in the graph.

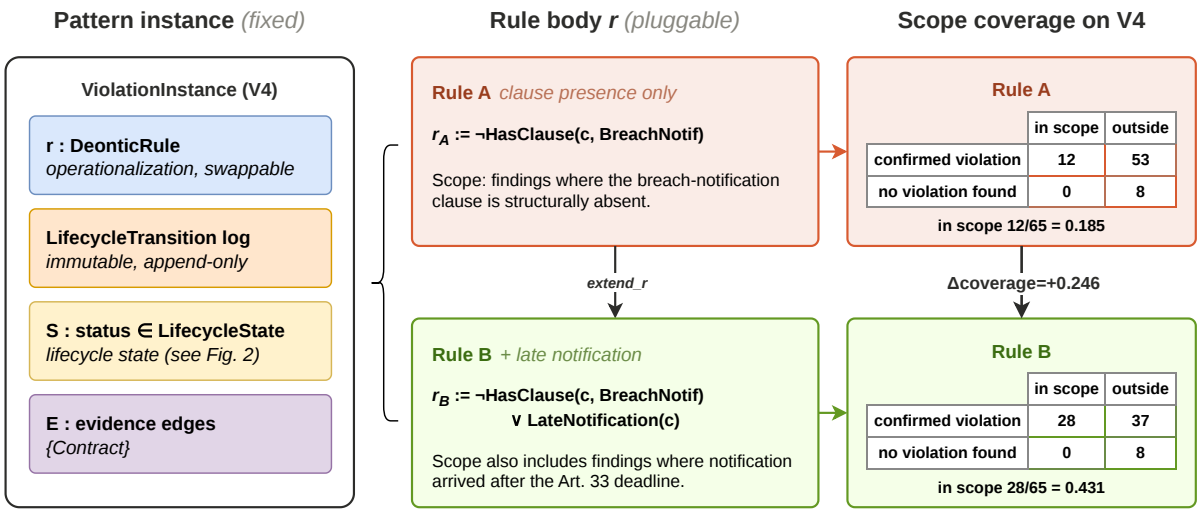


Figure 3: Rule-body scope coverage on V4. The pattern instance (left, fixed) is unchanged across two rule bodies (center, pluggable), whose scopes the figure states. Coverage on 73 GDPRhub enforcement decisions (right) rises from 0.185 to 0.431 when r is extended, with the pattern's identity, lifecycle, and evidence semantics unchanged.

Ground-truth construction. The synthetic V4 set is circular by construction, so we externalize it against GDPRhub [27]: all 92 non-subcategory cases tagged Article 33 were fetched via the MediaWiki API, and 73 yielded parseable outcomes. A transparent outcome-to-label mapping (*Violation Found*, *Upheld*, *Partly Upheld* to violation; *No Violation Found*, *Rejected*, *No further action* to non-violation; three parser-glitch outcomes hand-resolved) gives 65 violations and 8 non-violations. Subtype assignment combined keyword classification with manual verification of all 65 by DiliTrust compliance practitioners against the regulator's summary and Holding section, giving 12 missing-notification findings, 16 late-notification, and 37 other Article 33 findings that lie outside both rule bodies by design. Of the 16 late-notification cases, 8 carry specific hour values extracted with GPT-4.1 as a structured-extraction tool and verified by the same practitioners against the source documents; for the other 8 the regulator confirmed the delay exceeded 72 hours without stating a value, recorded as `above_threshold_unspecified` with the hour field null.

6.3. Portability: SHACL Cross-Formalism Check

The instantiation is a property graph because that is the stack the products run on. The check below tests whether the same semantics survive in an RDF stack, with both implementations operationalizing the same clause-presence condition and the success criterion being equal firing on equal scope. We export the Contract+Clause subgraph to RDF/Turtle (770 triples) and run pyshacl 0.31 against shapes encoding V3 and V4 with SHACL Core constraints [24] and targets expressed in SPARQL Protocol and RDF Query Language (SPARQL), using SHACL Advanced Features [28]. V1 and V2 are outside this check: their mandate-scope and signing-date conditions would require additional cross-subgraph constraints.

On identical scope the two formalizations fire on identical contract sets: V4 agrees on all ten data-processing agreements, and V3 agrees on all fifty contracts of the evaluation manifest. Widening V3's scope to the full contract set raises both implementations to the same 185 contracts, again with identical membership. The parity concerns detection alone. Persistent identity, lifecycle state, and audit traversal are reachable in an RDF stack only by adding custom vocabulary and orchestration on top of the standard.

7. Limitations, Threats to Validity, and Future Work

The limitations of this work fall into two groups: rule-level boundaries and system-level properties of the property-graph realization. We discuss each below alongside its natural extension.

7.1. Rule-Level Limitations

- **Rule evolution.** The evaluation covers rule-body revisions under a stable identifier and identity scheme. Withdrawing a detector does not itself delete existing situations or mark them remediated. A replacement yielding the same composite key matches the existing situation but retains its original rule reference; a changed key creates a separate record. The key alone cannot establish semantic continuity between different rules. Explicit rule-version provenance and successor mappings remain future work.
- **V4 detector boundary (Sections 5.2, 6.2).** None of the 37 findings outside Rule B's scope is reachable by a clause-presence or deadline predicate; reaching them requires clause-text natural-language inference and structured Article 33(5) predicates, detector extensions that keep the pattern unchanged.
- **V3 normative status (Section 5.2).** A `deonticForce` field on the rule would distinguish policy-derived from statutory obligations without altering the pattern.

7.2. System-Level Limitations

- **Observation granularity.** The transition log records lifecycle changes, not every positive rule match. A future extension could attach timestamped detection observations to the existing violation node, preserving intermediate re-detections without changing situation identity or the lifecycle automaton.
- **Upstream extraction quality.** Precision depends on upstream extraction (contract-signatory, clause-boundary, clause-type), which we do not separately benchmark; a full system evaluation would compose extraction-stage and pattern-stage error rates.
- **Application-layer enforcement (Section 4.2).** A caller bypassing the `transition()` mediator can violate either invariant, or both, so a deployment must confine write access to ViolationInstance and LifecycleTransition to the mediator itself; the guarantees are only as strong as that control. Future work will investigate declarative validation together with controlled, transactional updates for enforcing lifecycle invariants, using SHACL and an RDF representation [24, 25].
- **Operational properties.** Concurrent transitions and production-scale (10^5–10^7 instances) traversal performance are not evaluated. An optimistic check on `lastVerifiedAt` would serialize concurrent writes, and indexing `transitionedAt` and `violationId` is the expected scaling lever for the linearly growing transition log.

7.3. Threats to Validity

The rules are author-constructed. Practitioners confirmed the competency questions, but without formal elicitation or a structured survey; they reflect operational queries rather than systematic requirements analysis. Detection counts describe the curated manifest partition; stability checks also cover the full graph.

Two stability properties hold by construction; their checks verify implementation, while rule revision tests identity and history preservation. A deterministic exerciser and invariant test establish lifecycle enforcement, not suitability for real reviews.

The evaluation uses a production-schema fixture populated with public contract text and planted violations. Mandates, signing authorities, and roughly half the signing dates are synthetic; V3 and V4 use real text and clause annotations. The results concern pattern behaviour, not detection quality on real governance data. The artifact's GDPR-consent schema-transfer exercise is illustrative; systematic cross-domain validation remains future work.

Compliance practitioners independent of rule-body design supplied the coverage subtype labels, but no inter-rater agreement was measured. An independent annotation pass is needed to establish their reliability.

7.4. Future Work: Multi-Agent Compliance Workflows

We will investigate whether VSP as shared persistent memory improves coordination among agents detecting, reviewing, and re-evaluating findings across document revisions and overlapping evidence. Comparison with a structured issue store supporting deduplication and event history would measure duplicate findings, false merges, continuity, and stale or contradictory actions. Ablations would isolate identity, observation history, and lifecycle controls. Timestamped agent observations could supplement lifecycle transitions without changing situation identity. Practitioner judgments of finding identity and appropriate actions on realistically reconstructed cases would assess operational usefulness beyond implementation checks.

8. Conclusion

VSP represents compliance violations as persistent graph entities with composite identities, a five-state lifecycle, append-only transition histories, and evidence links. These support graph queries about first

detection, review decisions, and recorded re-openings. In the tested scenarios, repeated execution created no duplicate violations, and scope expansion preserved existing identities and transition histories while adding newly in-scope findings.

Future work will investigate shared violation memory for multi-agent compliance workflows, practitioner validation, and declarative validation combined with transactional lifecycle controls. Broader governance settings, including TAIR [17] and ARGOS [18], provide further contexts for assessing the pattern's generality.

Acknowledgments

All research contributions, methodological decisions, claims, and analyses are the authors' own. The authors have no competing interests to declare that are relevant to the content of this article.

Declaration on Generative AI

During the preparation of this work, the authors used GPT-4.1 in order to: extract deadline values from regulator case summaries (structured extraction, Section 6.2); and Anthropic's Claude in order to: grammar and spelling check, and rephrasing (editorial assistance). After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] R. David, A. Ahmeti, G. Bushati, A. Tauqeer, A. Fensel, A SHACL-based data consistency solution for contract compliance verification, in: ESWC 2025 Workshops and Tutorials Joint Proceedings, volume 3977 of *CEUR Workshop Proceedings*, 2025. URL: https://ceur-ws.org/Vol-3977/OPAL2025-1.pdf.

[2] L. Robaldo, C. Bartolini, M. Palmirani, A. Rossi, M. Martoni, G. Lenzini, Formalizing GDPR provisions in reified I/O logic: The DAPRECO knowledge base, Journal of Logic, Language and Information 29 (2020) 401–449. doi:10.1007/s10849-019-09309-z.

[3] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutierrez, S. Kirrane, J. E. Labra Gayo, R. Navigli, S. Neumaier, A.-C. Ngonga Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, A. Zimmermann, Knowledge graphs, ACM Computing Surveys 54 (2021) 71:1–71:37. doi:10.1145/3447772.

[4] E. Filtz, S. Kirrane, A. Polleres, The linked legal data landscape: Linking legal data across different countries, Artificial Intelligence and Law 29 (2021) 485–539. doi:10.1007/s10506-021-09282-8.

[5] V. Leone, L. Di Caro, S. Villata, Taking stock of legal ontologies: A feature-based comparative analysis, Artificial Intelligence and Law 28 (2020) 207–235. doi:10.1007/s10506-019-09252-1.

[6] A. Gangemi, P. Mika, Understanding the semantic web through descriptions and situations, in: On The Move to Meaningful Internet Systems 2003 (OTM 2003), volume 2888 of *LNCS*, Springer, 2003, pp. 689–706.

[7] N. Racz, E. Weippl, A. Seufert, A frame of reference for research of integrated governance, risk and compliance (GRC), in: Communications and Multimedia Security (CMS 2010), volume 6109 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 106–117. doi:10.1007/978-3-642-13241-4_11.

[8] M. Hashmi, G. Governatori, H.-P. Lam, M. T. Wynn, Are we done with business process compliance: state of the art and challenges ahead, Knowledge and Information Systems 57 (2018) 79–133. doi:10.1007/s10115-017-1142-1.

[9] N. S. Abdullah, M. Indulska, S. W. Sadiq, Compliance management ontology – a shared conceptualization for research and practice in compliance management, Information Systems Frontiers 18 (2016) 995–1020. doi:10.1007/s10796-016-9631-4.

[10] D. Hendrycks, C. Burns, A. Chen, S. Ball, CUAD: An expert-annotated NLP dataset for legal contract review, in: Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks, volume 1, 2021.
[11] Y. Koreeda, C. D. Manning, ContractNLI: A dataset for document-level natural language inference for contracts, in: Findings of the Association for Computational Linguistics (EMNLP 2021), 2021, pp. 1907–1919. doi:10.18653/v1/2021.findings-emnlp.164.
[12] G. Governatori, Z. Milosevic, A formal analysis of a business contract language, International Journal of Cooperative Information Systems 15 (2006) 659–685. doi:10.1142/S0218843006001529.
[13] G. Governatori, Z. Milosevic, S. Sadiq, Compliance checking between business processes and business contracts, in: Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2006), 2006, pp. 221–232. doi:10.1109/EDOC.2006.22.
[14] V. A. Carriero, A. Gangemi, A. G. Nuzzolese, V. Presutti, An ontology design pattern for representing recurrent situations, in: Advances in Pattern-Based Ontology Engineering, volume 51 of *Studies on the Semantic Web*, IOS Press, 2021, pp. 166–182. doi:10.3233/SSW210013.
[15] L. Waltersdorfer, M. Sabou, Leveraging knowledge graphs for AI system auditing and transparency, Journal of Web Semantics 84 (2025) 100849. doi:10.1016/j.websem.2024.100849.
[16] H. Dibowski, Full traceability and provenance for knowledge graphs, in: Formal Ontology in Information Systems (FOIS 2024), volume 394 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2024, pp. 223–237. doi:10.3233/FAIA241309.
[17] J. Hernandez, D. Golpayegani, D. Lewis, An open knowledge graph-based approach for mapping concepts and requirements between the EU AI act and international standards, AI and Ethics 5 (2025) 4463–4474. doi:10.1007/s43681-025-00708-6.
[18] N. D. Pathirage, O. Seneviratne, D. L. McGuinness, ARGOS: Ontology design patterns for governing dynamic data operations in LLM-powered applications, in: Joint Proceedings of WOP 2025 and HAIBRIDGE 2025, volume 4093 of *CEUR Workshop Proceedings*, CEUR-WS.org, Nara, Japan, 2025, pp. 45–56. URL: https://ceur-ws.org/Vol-4093/paper4.pdf.
[19] M. Overeem, M. Spoor, S. Jansen, The dark side of event sourcing: Managing data conversion, in: IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER 2017), 2017, pp. 193–204. doi:10.1109/SANER.2017.7884621.
[20] A. Krisnadhi, P. Hitzler, A core pattern for events, in: Advances in Ontology Design and Patterns, volume 32 of *Studies on the Semantic Web*, IOS Press, 2017, pp. 29–37. doi:10.3233/978-1-61499-826-6-29.
[21] V. Presutti, A. Gangemi, Content ontology design patterns as practical building blocks for web ontologies, in: Conceptual Modeling - ER 2008, volume 5231 of *LNCS*, Springer, 2008, pp. 128–141. doi:10.1007/978-3-540-87877-3_11.
[22] N. Noy, A. Rector, Defining N-ary relations on the semantic web, W3C Working Group Note, 2006. URL: https://www.w3.org/TR/swbp-n-aryRelations/.
[23] T. Lebo, S. Sahoo, D. McGuinness, PROV-O: The PROV ontology, W3C Recommendation, 2013. https://www.w3.org/TR/prov-o/.
[24] H. Knublauch, D. Kontokostas, Shapes constraint language (SHACL), W3C Recommendation, 2017. https://www.w3.org/TR/shacl/.
[25] R. Cyganiak, D. Wood, M. Lanthaler, RDF 1.1 concepts and abstract syntax, W3C Recommendation, 2014. URL: https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.
[26] European Commission, Commission implementing decision (EU) 2021/915 — standard contractual clauses, Official Journal of the European Union, 2021. URL: https://eur-lex.europa.eu/eli/dec_impl/2021/915/oj.
[27] GDPRhub, GDPRhub: A wiki of GDPR enforcement decisions, https://GDPRhub.eu/, 2025.
[28] H. Knublauch, D. Allemang, S. Steyskal, SHACL advanced features, W3C Working Group Note, 2017. URL: https://www.w3.org/TR/2017/NOTE-shacl-af-20170608/.