The Decision Event Model: Content-Addressed Records of Engineering Decisions

kgai maintainers

kgai.dev · team@kgai.dev · August 2026

ABSTRACT

Engineering teams routinely lose the reasoning behind their own systems. Architecture decision records preserve some of it as prose, but prose has no machine-readable structure, no stable identity, and no defined behavior under concurrent authorship. We describe the decision event model used by kgai, an open-source system that records engineering decisions as immutable, content-addressed events in an append-only log. Each event carries a title, a rationale, an author, a date, optional external references, the decisions it supersedes, and a list of structural mutations that reshape a small graph of domain elements such as features, services, and business objects. A decision's identity is a cryptographic hash of its content, which makes ingestion idempotent and makes every change a new recorded decision rather than an edit. Element identity is a deterministic hash of a normalized kind and name, so independent writers converge on the same node without coordination. We argue that recording decisions as events rather than as mutable state is what preserves rejected alternatives, supersession chains, and the ability to reconstruct any past state.

1. Introduction

The motivation behind an engineering decision is among the least durable artifacts a software team produces. The code that resulted from a decision survives in version control, but the reasoning, the alternatives that were weighed, and the approaches that were tried and abandoned usually survive only in conversation. Nygard's architecture decision records [1] addressed this by proposing short, numbered prose documents committed alongside the code, and the practice has been widely adopted.

Prose records have three structural limits, however. They are written for human readers and carry no machine-readable statement of what the decision actually changed. They have no intrinsic identity, so nothing defines what happens when two people record the same decision, or when one person records it twice. And their relationship to the parts of the system they govern remains implicit in their text.

These limits matter more once the primary reader and writer of such records is a machine. kgai is a local-first, MIT-licensed system built on the premise that engineering decisions and knowledge should be captured immutably and historically, primarily for an AI coding agent to read from and write to. Its unit of

record is the decision event, an immutable, content-addressed event that both documents a decision and mutates a small graph of domain elements.

This report describes that model. Section 2 presents the event schema. Section 3 argues for domain elements rather than code symbols as the graph's nodes. Section 4 describes the mutation operations. Section 5 develops the content-addressed identity scheme. Section 6 argues for events over mutable state as the storage discipline. The ordering of events from multiple writers and the deterministic projection of the log into a graph are treated in companion reports in this series (TR-2026-02 and TR-2026-03).

2. The Decision Event

A decision event is a single immutable record with a small, fixed set of fields.

Title. A short statement of what was decided.

Rationale. The reasoning, in prose. This is the field the whole system exists to preserve, the answer to the question of why the system is the way it is.

Author. Who recorded the decision. The value is taken from the writer's version control identity, so records made during normal work carry the identity the team already maintains.

Date. When the decision was made. The field accepts calendar dates or RFC 3339 timestamps, and it may be backdated deliberately, because a team importing decisions it made in the past should record them under their true dates rather than the import date. The date is provenance for human readers rather than the ordering key of the log, for reasons developed in the ordering report in this series (TR-2026-02).

References. Optional pointers to external artifacts, such as issue or ticket identifiers, when a team's conventions call for them.

Supersession. The prior decisions this decision replaces. A decision that reverses or refines an earlier one supersedes it, and the superseded decision remains in the record, queryable, marked as no longer in force.

Mutations. A list of structural operations on the element graph, described in Section 4. The mutations are the machine-readable content of the decision. They state exactly which elements the decision touched and how.

Events are stored as newline-delimited JSON, one event per line, in an append-only shard per writing installation. The format is deliberately plain. A store is readable, searchable, and portable without any of the system's own code, which we consider a survivability property for a record intended to outlive tooling choices.

Two absences from the schema are as deliberate as its contents. There is no free-form body that can be edited later, and there is no status field that is flipped in place when a decision is replaced. Either would

reintroduce mutation into a record whose value depends on immutability. Replacement is expressed only by a new event that supersedes the old one.

3. Elements as Domain Concepts

The nodes of the graph that decisions mutate are domain elements, application- and business-level things such as a feature, a service, or a business object. Each carries a kind and a name, for example a feature named product search or a business object named Invoice. They are deliberately few and stable.

The alternative, anchoring decisions to code symbols such as files, functions, or classes, we reject for a reason that follows from the record's purpose. Code symbols churn. Files are renamed, functions are split, classes migrate between modules, and a decision anchored to them decays as the code is refactored, even while the decision itself still governs the system. Domain concepts have much longer lifetimes. The invoice, the checkout, and the search feature persist across rewrites of the code that implements them. This mirrors the argument from domain-driven design that a stable model of the domain, held in a shared language, is the durable structure of a software system [3].

Keeping the element set small has a second consequence. The live graph, which represents only the current shape of the system, stays small enough to be discarded and rebuilt from the log cheaply. The projection report in this series (TR-2026-03) depends on that property.

4. Mutations

A decision carries a list of structural operations, applied atomically and idempotently as one unit. Four operations exist.

upsert_element introduces an element by kind and name, or resolves to the existing element if one is already present.

add_link adds a typed edge between two elements, for example a dependency of a feature on a service.

retire_link removes a live edge.

set_prop writes a key in a small property blob attached to an element, with keys kept sorted so the stored form is canonical.

The operations mutate only the live graph, the current shape. Nothing is lost when retire_link deletes an edge or set_prop overwrites a value, because the decision that made the change is itself permanent, and replaying the log reconstructs any past state on demand.

After its mutations are applied, a decision is connected to every element it touched by a SHAPES relationship, and it supersedes the prior head decision of every element it structurally changed. The result

is two planes over one body of data. The live plane holds elements and their current links. The decision plane holds the full immutable history, so the question of how an element came to its present shape is answered by reading the ordered decisions that shape it, each with its rationale, without any replay.

5. Content-Addressed Identity

Both identities in the system are deterministic functions of content, a discipline familiar from the content-addressed object store inside version control systems such as Git [5] and, in its authenticated form, from Merkle's hash-based signature constructions [4].

A decision's identifier is a cryptographic hash of its content. Three properties follow. First, ingestion is idempotent. Recording identical content twice produces the same identifier, and the second write is a no-op rather than a duplicate, which matters in a system whose writers include agents that may retry. Second, immutability is enforced by construction rather than by policy. No operation edits a decision, and any change to a decision's content is by definition a different decision with a different identifier. Third, records are verifiable. When events are transferred between machines during synchronization, each is checked against its hash on arrival.

An element's identifier is a hash of its normalized kind and name, and it deliberately excludes the author and the time of creation. Two writers who independently record the feature named product search therefore mint the same identifier, and the projection merges their contributions into a single node. Convergence on shared vocabulary requires no coordination, no central registry of elements, and no reconciliation step. Deduplication is a property of the identity scheme rather than a background process.

The cost of this choice is that identity is exactly as good as the normalization. Two teams that name one concept two ways create two elements. We return to this in Section 7.

6. Why Events Rather Than State

A system with the same goals could store the current description of each element and update it in place. We store events instead, in the tradition of event sourcing [2], and each reason is visible as a concrete capability.

First, a state store answers only in the present tense. An event log preserves every state. The system can replay every decision recorded at or before a given time into an ephemeral graph and answer what the system looked like on any past date. Combined with honest backdating on import, this makes the timeline a faithful record rather than an artifact of when the tooling was adopted.

Second, updating in place destroys precisely the records with the highest value, the rejected alternatives. When a team hides sold out products from search and then reverses that choice because of what it cost, a state store retains only the current behavior. The event log retains the original decision, the reversal, and

the reason for each. Dead ends stay queryable together with why they were abandoned, which is what prevents a future engineer, or a future agent, from walking the same path again.

Third, immutable append-only data has favorable distribution properties, an observation Helland develops in general form [6]. Because events are never modified and each installation appends only to its own shard, replicas exchange events without any merge logic over content, and no two writers can ever collide on a file. Genuine contradictions, where two decisions concurrently claim to be the current word on one element, surface explicitly as two competing head decisions, and are resolved by recording one new decision that supersedes both. History retains the branch and its resolution alike. Conflict is thereby promoted from a storage accident to a first class, auditable event in the record.

Fourth, an event log composes naturally with derived read models [7]. The graph an agent queries is a projection of the log and can be rebuilt from it at any time. The projection report in this series (TR-2026-03) treats this in full.

7. Limitations

The record is only as complete as its writers. A decision that was never recorded cannot be recovered from the log, and the quality of a rationale is fixed at write time by whoever writes it. The model preserves reasoning. It cannot create it.

Element identity is nominal. Because the identity hash converges on normalized kind and name, the same concept under two names becomes two elements, and unifying them is itself a modeled change rather than something the system infers.

Granularity is a modeling choice. The model gives no rule for what deserves to be an element. Too fine, and the graph inherits the churn of code symbols that Section 3 argues against. Too coarse, and decisions lose precision about what they touched.

Finally, the schema is shaped for structural decisions about a system. Decisions that touch no element still fit the log, but the model's leverage comes from mutations, supersession, and the element graph, and it offers less to knowledge that has no structural footprint.

8. Conclusion

The decision event model treats an engineering decision as data with three obligations that prose does not meet. It must state, in machine-readable form, what it changed. It must have an identity that makes duplication harmless and editing impossible. And it must remain in the record when it is overturned, connected to what replaced it. Content addressed events carrying structural mutations over a small graph of domain elements meet all three, and they do so with an append-only log of plain text as the entire storage commitment. Companion reports in this series describe how logs from many writers are ordered

without coordination (TR-2026-02) and how the element graph is derived from the log deterministically (TR-2026-03).

References

[1] M. Nygard. 2011. Documenting Architecture Decisions. Cognitect blog. https://www.cognitect.com/blog/2011/11/15/documenting-architecture-decisions

[2] M. Fowler. 2005. Event Sourcing. martinfowler.com. https://martinfowler.com/eaaDev/EventSourcing.html

[3] E. Evans. 2003. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley.

[4] R. C. Merkle. 1987. A Digital Signature Based on a Conventional Encryption Function. Advances in Cryptology, CRYPTO '87, Lecture Notes in Computer Science, Vol. 293. Springer, 369-378.

[5] S. Chacon and B. Straub. 2014. Pro Git, Second Edition. Apress.

[6] P. Helland. 2015. Immutability Changes Everything. ACM Queue 13(9). https://queue.acm.org/detail.cfm?id=2884038

[7] M. Kleppmann. 2017. Designing Data-Intensive Applications. O'Reilly Media.