Decision Telemetry Architecture
A Universal Contract for Analytics-Ready Systems

Shobha Sethuraman

✦

Executive Summary

Modern analytics teams spend enormous effort reconstructing decisions from logs, events, and scattered artifacts. This reactive model turns analytics into archaeology: pipelines compensate for upstream ambiguity, investigations rely on reverse engineering, and explainability becomes expensive.

This paper argues that analytics maturity requires a structural shift. Decisions must be emitted as first-class semantic objects at runtime rather than inferred after the fact. We introduce decision contracts as a standardized representation of outcomes, evidence, and lineage, and describe an architecture pattern that makes analytics-ready systems intentional by design.

Decision telemetry does not replace existing observability or data contracts. It adds a semantic layer that allows analytics systems to consume declared intent instead of reconstructing behavior. The result is faster investigations, clearer audit trails, and predictable analytics pipelines.

This paper presents a maturity model, architectural framework, and reference implementation demonstrating how decision contracts can function as a universal standard for analytics-driven systems.

Abstract

Modern analytics systems operate in a fundamentally reactive mode: they reconstruct decisions after the fact from logs, events, and partial artifacts. This creates fragile pipelines, high investigation cost, and limited explainability in domains such as fraud detection, compliance, and AI-driven systems. While data contracts have improved structural stability, they address schema integrity rather than semantic intent. This paper argues that decisions themselves must become first-class architectural objects.

We introduce the concept of *decision contracts*: a standardized runtime representation of outcomes, evidence, and lineage that is emitted at the moment a decision occurs. Decision contracts allow analytics systems to consume intent directly rather than infer it indirectly. We propose a three-stage analytics maturity model - reactive cleanup, structural contracts, and intentional decision instrumentation - and describe an architecture pattern intended to enable analytics-ready systems by design. A reference implementation demonstrates the practicality of this approach and suggests a path toward a universal decision telemetry standard.

1. The Hidden Tax of Reactive Analytics

Most analytics systems today operate in a reactive mode. Producer systems emit logs, events, and state changes without an explicit representation of intent. Analytics teams then reconstruct decisions by correlating artifacts across multiple sources. Fraud investigations, operational debugging, and compliance analysis often require reverse-engineering why a system behaved the way it did.

This pattern treats analytics as a downstream cleanup function. Pipelines become infrastructure dedicated to interpretation rather than insight. Engineers maintain fragile transformations that depend on implicit assumptions about upstream behavior. When upstream systems change, analytics silently degrades or requires costly rework. The organization accumulates what can be described as a semantic debt: intent exists in code, but not in data.

This reactive model does not fail because of insufficient tooling. It fails because producer systems are not designed to emit decision semantics. Logs capture behavior; metrics capture aggregates; events capture transitions. None of these primitives guarantee an explicit statement of why a decision occurred.

As systems grow more automated and AI-driven, this gap widens. The cost of reconstructing decisions increases faster than the value of analytics derived from them. A structural shift is required: decisions must be observable by design, not inferred after the fact.

1.1 The Urgency of Now

While the emerging discourse on **Context Graphs** and the **"Two Clocks Problem"** has recently amplified the need for capturing decision-logic in agentic systems [8], this gap has long been a fundamental bottleneck in the analytics world. For decades, analytics teams have been forced into "forensic archaeology", attempting to reconstruct intent from fragmented logs and state changes. **Decision Telemetry** addresses this historical debt by providing a universal architecture that ensures intent is declared as a first-class object at runtime rather than inferred after the fact.

This technical debt has now evolved into a **strategic risk** driven by three converging trends:

1. **Shift to Probabilistic Systems**: AI agents and LLMs produce outputs that cannot be audited by reading code alone; understanding these actions

requires an immutable trace of the context and logic used at the moment of execution.
2. **Complexity of Automated Pipelines**: Modern workflows, such as fraud detection, are complex DAGs of rules and ML models. Debugging a single outcome now requires tracing a path through dozens of independent, opaque components.
3. **Regulatory Mandates for Explainability**: New frameworks like the EU AI Act and financial compliance regulations demand more than just logs; they require a mechanical proof of *why* a decision was made [3]. Organizations can no longer rely on downstream reconstruction to meet legal evidentiary standards.

The next section examines why structural data contracts alone cannot resolve this limitation [2].

2. Limits of Structural Data Contracts

The introduction of data contracts represents an important maturity step for analytics systems [2]. By establishing explicit schema guarantees, organizations reduce accidental breakage and improve producer accountability. Structural contracts create predictability in data pipelines and prevent downstream failures caused by undocumented changes.

However, structural stability is not equivalent to semantic clarity.

A data contract answers the question:

What fields exist?

It does not answer:

What decision happened and why?

Even perfectly stable schemas can encode ambiguous intent. Logs may be well-structured yet still require interpretation. Metrics may be consistent yet detached from the reasoning that produced them. Events may capture transitions without declaring the decision logic that drove those transitions.

Contracts prevent damage; they do not add meaning.

This distinction is critical. Structural contracts stabilize pipelines, but they do not eliminate the need for downstream inference. Analytics teams still reconstruct intent indirectly, using domain knowledge and heuristics. The system becomes safer, but not more explainable.

The next stage of maturity requires contracts that express semantics, not only structure.

3. Decisions as First-Class Objects

All software systems that interact with the real world are decision systems. They classify, approve, reject, route, score, escalate, and prioritize. These decisions already exist as executable logic, but they rarely exist as explicit data objects.

A decision can be defined as a comprehensive semantic object composed of five core elements:

Decision = Context + Actor + Logic + Outcome + Lineage

1. **Context**: The environmental state (Timestamp, System Version, Environment).
2. **Actor**: The entity responsible for the decision (System, AI Agent, or Human).
3. **Logic**: The evidence used and policies evaluated (Rules, Model Scores, Checks).
4. **Outcome**: The final result (Approved, Rejected, Flagged).
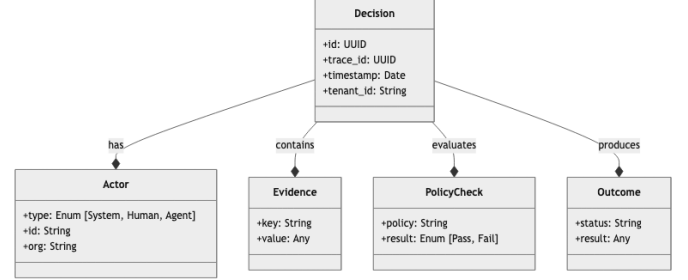5. **Lineage**: The causal chain of preceding decisions (Parent/Child relationships).

Figure 1. Decision Object Diagram

When these elements are emitted intentionally at runtime, the system declares its reasoning rather than forcing analytics to infer it.

A *decision contract* is the standardized representation of this object. It is not a logging convention. It is a semantic declaration that a meaningful decision has occurred, recorded in an append-only ledger.

This shift transforms decisions from implicit side effects into explicit artifacts. Analytics systems no longer reverse engineer intent; they consume it directly. Investigations become queries against declared reasoning rather than forensic reconstruction.

3.1 Anatomy of a Decision Event

To make this concrete, a decision is not a single log line but a structured event containing specific fields. The schema is language-agnostic and transport-neutral. Below is a JSON representation of a decision outcome event as defined in the reference implementation:

```
{
  "event_type": "decision.outcome",
  "decision_id": "dec_01HQK4K...",
  "trace_id": "trc_01HQK4L...",
  "timestamp": "2026-02-09T14:30:00Z",
  "actor": {
    "type": "system",
    "id": "risk_engine_v2"
  },
  "payload": {
    "status": "approved",
    "risk_score": 0.05
  },
  "causal_links": [
    {
      "type": "triggered_by",
      "target_decision_id": "dec_PREV..."
    }
  ]
}
```

This strict schema ensures that every decision - whether from a microservice, a monolith, or an AI agent - looks identical to the analytics layer.

3.2 Safety and Privacy by Design

A critical requirement for decision telemetry is safety. Recording decisions must never compromise the system's stability or the user's privacy.

3.2.1 PII Redaction

Decision payloads often contain sensitive data (names, emails, tokens). The telemetry layer must support automatic redaction of Personally Identifiable Information (PII) at the source, before the event leaves the producer system.

3.2.2 Failure Isolation

Telemetry is a non-critical path. If the decision recording infrastructure fails (e.g., the collector is down), the application must continue to function normally. The SDK isolates telemetry errors, ensuring that observability issues never cause business logic failures.

3.2.3 Immutability

Once a decision is recorded, it is written to an append-only ledger. This guarantees that the history of reasoning cannot be altered, providing a reliable audit trail for compliance and governance.

4. Analytics Maturity Model

Analytics architecture evolves through recognizable stages of maturity. These stages are not mutually exclusive; they represent an accumulation of capabilities.
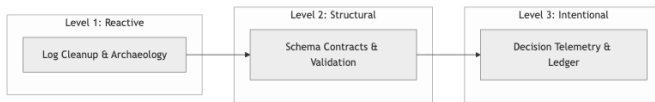

Figure 2. Analytics Maturity Model

4.1 Level 1: Reactive Cleanup

At this stage, analytics reconstructs decisions from artifacts. Pipelines compensate for inconsistent upstream emissions. Knowledge of system behavior is embedded in transformation logic and institutional memory. Analytics functions as archaeology.

4.2 Level 2: Structural Contracts

Here, organizations introduce schema guarantees and producer accountability. Pipelines become stable. Breakage decreases. However, intent remains implicit. Analytics consumes structured artifacts but still performs semantic inference.

4.3 Level 3: Intentional Decision Telemetry

In the final stage, decisions are emitted as structured semantic objects. Producers instrument decision contracts directly. Analytics consumes declared intent rather than reconstructing it. Explainability, auditability, and investigation speed improve as a consequence of architecture, not heroics.

This model does not replace earlier stages. Structural contracts remain essential. Decision telemetry builds on that foundation by adding meaning to structure.

5. Decision Telemetry Architecture Pattern

Decision Telemetry Architecture introduces a dedicated semantic layer between producer systems and analytics consumers.

1. **Producer Systems**: Application code instruments decision points using an SDK. At the moment a decision occurs, it emits a structured event (Start, Evidence, Policy Check, Outcome).
2. **Collector**: A lightweight agent receives these events, validates them against the schema, and buffers them for efficient transmission.
3. **Decision Ledger**: The core of the architecture is the **Append-Only Ledger**. Unlike mutable databases, the ledger stores the immutable history of decisions. The ledger is a logical architectural concept, not a specific storage technology. It guarantees that reasoning can be audited exactly as it happened.
4. **Analytics Consumers**: Downstream systems (Fraud Detections, Compliance Audits, AI Governance) consume from the ledger. They query declared intent rather than inferring behavior from logs.
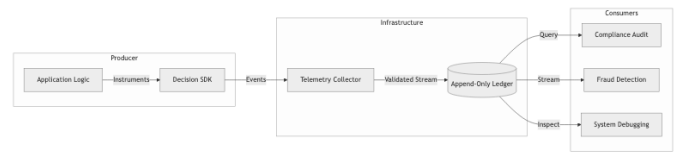

Figure 3. Decision Telemetry Architecture Flow

This architecture resembles the evolution of observability systems. Metrics and traces did not replace logs; they added a higher-level abstraction that made system behavior legible. Decision telemetry performs the same function for *intent*.

The pattern is incremental. Systems can instrument high-value decision points first, expanding coverage over time. Adoption does not require rewriting existing architecture; it requires declaring semantics where they already exist.

6. Production Profile: Safe Decision Telemetry in High-Throughput Systems

Decision telemetry is designed to capture reasoning at runtime without compromising system stability. In high-throughput environments, including payment gateways, fraud pipelines, AI inference services, observability infrastructure must obey strict performance and safety constraints. A telemetry system that introduces latency or memory pressure becomes an outage vector.

This section defines the production profile of decision telemetry: the operational rules required to ensure that decision recording remains safe under extreme load.

> The core principle is simple: decision telemetry must never be able to take down the host system.

All implementation guidance follows from this invariant.

6.1 Non-Blocking Capture

Decision recording occurs in the application hot path. Any blocking behavior - synchronous network calls, deep object copying, or expensive serialization - directly impacts user latency.

In production systems:

- decision capture must complete in constant time

- export must occur asynchronously
- application threads must never wait for telemetry I/O

The SDK performs only minimal validation and enqueues the event into a bounded in-memory buffer. A background worker handles serialization, batching, and transmission to the collector.

This separation ensures that telemetry overhead remains predictable even when downstream infrastructure experiences latency spikes or outages.

6.2 Bounded Memory and Backpressure

Telemetry systems fail not by silence but by explosion. Unbounded buffers allow observability traffic to accumulate until memory exhaustion destabilizes the host application.

The production profile requires:

- fixed-size ring buffers
- explicit payload size limits
- capped causal link arrays
- drop-on-overflow behavior

When the buffer is full, the oldest events are discarded and a counter metric is incremented. Dropping telemetry is acceptable. Blocking business logic is not.

This mirrors the design philosophy of mature observability systems: telemetry is best-effort, not transactional. Decision telemetry is intentionally best-effort. Preserving application reliability takes precedence over preserving telemetry.

6.3 Evidence Layering

A common failure mode in observability tooling is *payload dumping* - attaching entire application objects to telemetry events. Large mutable objects increase CPU overhead, create privacy risk, and amplify garbage collection pressure.

Decision telemetry enforces a layered evidence model:

1. **Layer 1 - Causal Core (mandatory)**: A minimal immutable record of the decision: identifiers, timestamp, actor, and outcome. This layer is intentionally small and always captured.
2. **Layer 2 - Context (bounded)**: Primitive key-value attributes describing environment or classification. Nested structures and large blobs are disallowed.
3. **Layer 3 - References (externalized state)**: Pointers to external systems (URIs, object store keys, warehouse IDs) rather than embedded data. This preserves reconstructability without copying large state into telemetry streams.

This model preserves semantic richness while maintaining predictable performance.

6.4 Fail-Open Isolation

Decision telemetry is non-critical infrastructure. Failures in the telemetry pipeline must never propagate to business logic.

If serialization fails, buffers overflow, or collectors are unavailable:

- events are dropped
- failure metrics are emitted
- application execution continues normally

The system degrades observability, not availability.

This isolation is essential in regulated or financial environments where reliability requirements exceed telemetry guarantees.

6.5 Transport and Schema Considerations

Production deployments may choose binary encodings (e.g., Protobuf or Avro) for efficiency, while retaining JSON as a human-readable debug format. The transport format is an implementation detail; the semantic contract remains constant.

Strict schemas stabilize ingestion and enable evolution through versioning. Producers emit decisions optimistically; collectors validate and normalize events before persistence in the append-only decision ledger.

The architecture favors loose coupling:

- producers focus on fast emission
- collectors enforce correctness
- ledgers guarantee immutability

6.6 Operational Envelope

A production-safe implementation targets:

- sub-millisecond p99 capture overhead
- sustained high event throughput per instance
- fixed upper memory bounds
- zero impact on request success rates

These constraints treat decision telemetry as *infrastructure*, not logging.

6.7 Anti-Patterns to Avoid

The following practices violate the production profile:

- synchronous HTTP export in request threads
- deep copying complex objects
- embedding full request or user objects in payloads
- unbounded arrays or recursive structures
- retry loops in the application hot path

These patterns convert observability into a reliability risk.

6.8 Why the Production Profile Matters

Decision telemetry's value lies in making reasoning observable. That value disappears if the instrumentation itself destabilizes the system it observes.

By enforcing non-blocking capture, bounded memory, layered evidence, and fail-open isolation, decision telemetry becomes safe to deploy in the environments where explainability matters most.

This production profile ensures that intentional analytics does not come at the cost of operational safety.

Telemetry remains best-effort. Architecture remains reliable.

And the system remains legible.

The production profile follows principles established in resilient systems design literature. Patterns such as non-blocking capture and bounded buffers are widely recommended for production software to prevent observability systems from amplifying failure conditions. Reliability engineering practice likewise emphasizes that telemetry must degrade gracefully rather than compromise availability. Decision telemetry adopts these constraints as architectural invariants, aligning with established production safety guidance [5], [6].

7. Practical Implications

The practical impact of decision telemetry is most visible in domains where explainability is essential.

7.1 Fraud Systems

Fraud investigations often reconstruct rule chains and model reasoning from scattered artifacts. Decision contracts collapse this process into a single semantic object. Investigation time decreases, and rule tuning becomes data-driven rather than inferential.

7.2 AI Governance

AI-driven systems face increasing demands for explainability. Decision telemetry provides a structured audit trail of reasoning inputs and outputs. Governance shifts from post-hoc interpretation to runtime transparency.

7.3 Compliance and Audit

Regulatory environments require reproducible explanations. Decision contracts encode reasoning as data, enabling mechanical audit rather than manual reconstruction.

7.4 Debugging Distributed Systems

Complex systems fail through cascades of decisions. Lineage-aware decision telemetry reveals causal chains that logs alone obscure.

In each case, the benefit is not new analytics capability. It is the removal of interpretive friction.

8. Reference Implementation

The Decision Trace SDK provides a reference implementation of decision telemetry concepts [7]. It demonstrates that decision contracts can be emitted with minimal overhead and integrated incrementally into existing systems.

The implementation is not the architecture. It is evidence that the architecture is practical.

Multiple implementations could exist. The value lies in the contract, not the tool.

9. Related Work

Decision telemetry does not exist in a vacuum. It builds upon and extends several established disciplines:

- **Structured Logging**: Provides the format (JSON) but lacks the semantic schema for decisions. Logs tell you *what* happened, not *why* [4].

- **OpenTelemetry (OTel)**: Solves the distributed context propagation problem (traces/spans) but focuses on *latency* and *errors*. Decision telemetry leverages OTel's context propagation but adds a semantic layer for *business logic* [1].
- **Data Contracts**: Enforce schema stability for analytics but often miss the diverse runtime context of a decision. Decision contracts are a specialized form of data contract focused on reasoning [2].
- **Event Sourcing**: Records state changes. Decisions are often the *cause* of state changes. Recording the decision (the "why") complements the event (the "what") [4].
- **Audit Trails**: Traditionally separate, manual, and partial. Decision telemetry makes audit trails a natural byproduct of the system's operation [6].
- **Explainable AI (XAI)**: Focuses on model weights and features. Decision telemetry captures the *system* context around the model, including policy overrides and human-in-the-loop actions [3].

Synthesis: Decision telemetry extends these inputs by treating decisions as **semantic contracts**. It bridges the gap between low-level infrastructure observability (OTel) and high-level business analytics (Data Contracts).

10. Toward a Universal Standard

For decision telemetry to reach its full potential, it must move beyond agent-specific memory models. While the rise of autonomous agents has highlighted the urgency of capturing reasoning traces, the requirement for an immutable, semantic record of intent is universal across all modern software. **Decision Telemetry** provides the standardized vocabulary required to bridge legacy microservices, manual human approvals, and autonomous agents into a single, analytics-ready governance layer.

A universal schema allows interoperability across systems, organizations, and industries. The path forward resembles earlier infrastructure standards: open specification, reference implementations, and community-driven evolution. As adoption grows, decision telemetry can serve as a fundamental layer, similar to metrics and tracing in modern systems [1].

Standardization does not constrain innovation. It creates a stable semantic layer on which innovation compounds.

11. Conclusion

Modern analytics spends enormous effort reconstructing intent from artifacts. This is not an inevitability of distributed systems. It is a consequence of architecture that ignores decisions as first-class objects.

Decision telemetry proposes a simple shift: emit reasoning at the moment it occurs. This shift aligns analytics systems with modern observability and reliability principles that treat runtime behavior as a first-class architectural concern [1], [6].

When decisions are declared explicitly, analytics stops being archaeology. It becomes architecture.

And architecture scales.

References

[1] OpenTelemetry Project, "OpenTelemetry Specification," v1.38.0, 2024. [Online]. Available: https://opentelemetry.io/docs/specs/otel/

[2] C. Sanderson, M. Freeman, and B. E. Schmidt, *Data Contracts: Developing Production-Grade Pipelines at Scale.* O'Reilly Media, 2025.

[3] European Union, "Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence," Official Journal of the European Union, 2024. [Online]. Available: http://data.europa.eu/eli/reg/2024/1689/oj

[4] M. Fowler, Event Sourcing, 2005. [Online]. Available: https://martinfowler.com/eaaDev/EventSourcing.html

[5] M. T. Nygard, *Release It!: Design and Deploy Production-Ready Software.* Pragmatic Bookshelf, 2018.

[6] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site Reliability Engineering: How Google Runs Production Systems.* O'Reilly Media, 2016. [Online]. Available: https://sre.google/books/

[7] S. Sethuraman, "Decision Trace SDK," GitHub repository, 2026. [Online]. Available: https://github.com/logicoflife/decision-trace

[8] J. Gupta and A. Garg, "AI's trillion-dollar opportunity: Context graphs," Foundation Capital, Dec. 22, 2025. [Online]. Available: https://foundationcapital.com/context-graphs