Algebraic Domain-Driven Design: A Categorical Semantics for Consistency in Healthcare Provider Directories

Carlos Vera-Ciro

Abstract

Healthcare provider directories integrate partially overlapping representations of the same real-world entities across multiple bounded contexts. Empirical audits report persistent inaccuracy rates exceeding 40% (Centers for Medicare and Medicaid Services, 2018; United States Government Accountability Office, 2018), largely due to non-deterministic entity resolution, lossy schema mappings, and temporal drift. This manuscript provides a categorical semantics for Domain-Driven Design (DDD) to address these systemic failures. Five structural results are proved: (1) entity resolution is a colimit construction in a category of keyed fragments; (2) Conflict-Free Replicated Data Type (CRDT) merge operations are computed by the join within a fixed semilattice, guaranteeing commutativity, associativity, and idempotency; (3) functorial schema translation preserves referential integrity for constraints expressible as pullback squares; (4) event-sourced aggregates form presheaves over time, with state reconstruction given by a natural transformation; and (5) the event-log presheaf satisfies the sheaf condition, enabling split-brain resolution via the gluing axiom. These constructions are validated with a companion TypeScript implementation using `fp-ts`, including property-based tests verifying the semilattice laws and presheaf functoriality. The framework is applied to a detailed case study of provider directory reconciliation under the No Surprises Act (United States Congress, 2022).

1 Introduction

Healthcare provider directories are fundamentally distributed data systems. A single provider's profile is rarely mastered in one place; rather, it is composed of partially overlapping fragments scattered across distinct bounded contexts (Evans, 2004). A clinical Electronic Health Record (EHR) system tracks physical clinic addresses, a credentialing database monitors medical licenses, and payer systems manage network enrollment.

Failures in directory accuracy arise at the boundaries of these contexts. A 2018 review by the Centers for Medicare and Medicaid Services found that over 40% of provider directory entries contained inaccuracies (Centers for Medicare and Medicaid Services, 2018), and the Government Accountability Office identified systemic deficiencies in CMS's oversight of directory accuracy (United States Government Accountability Office, 2018). The No Surprises Act (United States Congress, 2022), effective January 2022, imposes financial penalties on plans that maintain inaccurate directories, elevating data quality from an operational concern to a regulatory imperative.

When fragments cannot be deterministically merged, data is corrupted. When schema mappings discard structural information, relationships are orphaned. As network updates arrive out of order, and as queries demand time-indexed correctness (e.g., "Was this provider in-network yesterday?"), imperative integration scripts fail under the complexity. These are not engineering accidents to be patched with ad-hoc heuristics, but algebraic problems that demand a formal categorical semantics.

Contributions. This manuscript makes five contributions:

(i) Entity resolution is modeled as a colimit in the slice category $\mathbf{FinSet}/K$ (Section 3), yielding a deterministic, universal merge construction.

(ii) The relationship between Conflict-Free Replicated Data Type (CRDT) merge and join-semilattice structure is clarified, showing that the merge is the *internal* join rather than a categorical colimit in **JSL** (Section 5).

(iii) Safe schema translation is formalized via the adjoint triple $\Sigma_F \dashv \Delta_F \dashv \Pi_F$ and identify the pullback conditions under which referential integrity is preserved (Section 6).

(iv) Event-sourced aggregates are modeled as presheaves, with state reconstruction given by a natural transformation from the event-log presheaf to a state presheaf (Section 7).

(v) The event-log presheaf is shown to satisfy the sheaf condition, enabling global consistency via the gluing axiom and providing a formal mechanism for split-brain resolution (Section 8).

These constructions are validated by a companion TypeScript implementation (Section 9) and applied to a healthcare provider directory case study.

2 Background and Notation

This section briefly recalls the categorical and domain-driven design concepts used throughout the manuscript. Standard references for category theory include Mac Lane (Mac Lane, 1998) and Awodey (Awodey, 2010); for applied category theory, Fong and Spivak (Fong and Spivak, 2019).

2.1 Category Theory Preliminaries

Definition 2.1 (Category)**.** *A* category $\mathcal{C}$ *consists of a collection of objects, a collection of morphisms* $f : A \to B$ *between objects, an associative composition operation, and an identity morphism* id_A *for each object* A*.*

Definition 2.2 (Functor)**.** *A* functor $F : \mathcal{C} \to \mathcal{D}$ *maps objects and morphisms of* $\mathcal{C}$ *to those of* $\mathcal{D}$*, preserving composition and identities:* $F(g \circ f) = F(g) \circ F(f)$ *and* $F(\mathrm{id}_A) = \mathrm{id}_{F(A)}$*.*

Definition 2.3 (Natural Transformation)**.** *Given functors* $F, G : \mathcal{C} \to \mathcal{D}$*, a* natural transformation $\eta : F \Rightarrow G$ *assigns to each object* A *a morphism* $\eta_A : F(A) \to G(A)$ *such that for every morphism* $f : A \to B$*, the diagram*

$$\begin{array}{ccc} F(A) & \xrightarrow{\eta_A} & G(A) \\ {\scriptstyle F(f)}\downarrow & & \downarrow{\scriptstyle G(f)} \\ F(B) & \xrightarrow[\eta_B]{} & G(B) \end{array}$$

commutes.

Definition 2.4 (Limit and Colimit)**.** *Let* $D : J \to \mathcal{C}$ *be a diagram (a functor from a small indexing category* J*). A* limit *of* D *is a universal cone over* D*; a* colimit *is a universal cocone under* D*. Familiar special cases include products (limits over discrete diagrams), equalizers, pullbacks, coproducts, coequalizers, and pushouts (Mac Lane, 1998).*

Definition 2.5 (Adjunction)**.** *An* adjunction $F \dashv G$ *between categories* $\mathcal{C}$ *and* $\mathcal{D}$ *consists of functors* $F : \mathcal{C} \to \mathcal{D}$ *and* $G : \mathcal{D} \to \mathcal{C}$ *together with a natural bijection* $\mathrm{Hom}_{\mathcal{D}}(F(A), B) \cong \mathrm{Hom}_{\mathcal{C}}(A, G(B))$*. Right adjoints preserve limits; left adjoints preserve colimits (Mac Lane, 1998).*

Definition 2.6 (Presheaf)**.** *A* presheaf *on a category* $\mathcal{C}$ *is a functor* $P : \mathcal{C}^{op} \to \mathbf{Set}$*. The category of presheaves* $[\mathcal{C}^{op}, \mathbf{Set}]$ *is always complete and cocomplete (Mac Lane, 1998).*

2.2 Domain-Driven Design

Domain-Driven Design (Evans, 2004; Vernon, 2013) organizes complex software around a *ubiquitous language* grounded in the business domain. The key structural concepts are:

- **Bounded Context.** An explicit boundary within which a domain model is consistent. Different contexts may model the same real-world entity with different schemas.
- **Aggregate.** A cluster of domain objects treated as a single transactional unit, with a root entity enforcing invariants.
- **Domain Event.** An immutable record of something that happened in the domain, timestamped and ordered.
- **Anti-Corruption Layer (ACL).** A translation mechanism at a context boundary that prevents foreign models from leaking in.

Table 1 summarizes the correspondence between DDD concepts and their categorical counterparts used in this manuscript.

Table 1: DDD–Category Theory correspondence.

DDD Concept	**Categorical Construction**
Bounded Context	Category
Aggregate	Object
Domain Event	Morphism / element of presheaf $P(t)$
Anti-Corruption Layer	Functor between categories
Entity Resolution	Colimit in $\mathbf{FinSet}/K$
CRDT Merge	Join in a semilattice
Schema Translation	Adjoint triple $\Sigma_F \dashv \Delta_F \dashv \Pi_F$
Temporal State	Presheaf $P : T^{op} \to \mathbf{Set}$
Global Consistency	Sheaf condition (gluing axiom)

2.3 Healthcare Provider Directories

A healthcare provider directory aggregates data from at least four bounded contexts:

1. **Credentialing Context:** Source of truth for medical licenses, board certifications, and training history.
2. **Clinical / EHR Context:** Source of truth for physical clinic locations, daily schedules, and patient panels.
3. **Contracting / Billing Context:** Source of truth for insurance network participation and reimbursement rates.
4. **Public Directory Context:** The consumer-facing search engine, a read model composed from the above sources.

Each context maintains its own partial view of a provider identified by a National Provider Identifier (NPI). The public directory must reconcile these partial views into a single, accurate profile. Under the No Surprises Act (United States Congress, 2022), plans face penalties for publishing inaccurate directories, particularly regarding network status and office locations.

3 The Category of Keyed Fragments

To formalize fragmented data, a category is defined whose objects are the partial views held by individual bounded contexts.

Definition 3.1 (Keyed Fragment)**.** *Let K be a fixed finite set of global provider identifiers (e.g., NPI numbers). A* keyed fragment *is a pair (S, κ) where:*

- *S is a finite set of records,*
- *$\kappa : S \to K$ assigns each record to its global provider key.*

Definition 3.2 (Category of Keyed Fragments)**.** *The category $\mathbf{Frag}_K$ has:*

- ***Objects:*** *keyed fragments (S, κ),*
- ***Morphisms*** *$f : (S, \kappa) \to (T, \lambda)$: functions $f : S \to T$ preserving the global key, i.e., $\lambda \circ f = \kappa$.*

Lemma 3.3. *$\mathbf{Frag}_K$ is finitely complete and finitely cocomplete.*

Proof. $\mathbf{Frag}_K$ is equivalent to the slice category $\mathbf{FinSet}/K$. The category **FinSet** has all finite limits and finite colimits, and slice categories over a fixed object inherit finite (co)completeness (Mac Lane, 1998). □

Remark 3.4. *Finite (co)completeness suffices for present purposes because bounded context diagrams in practice are finite: a healthcare directory integrates data from a fixed, enumerable collection of source systems. The arbitrary (co)completeness of* **FinSet** *does not hold (e.g.,* **FinSet** *lacks infinite coproducts), but infinite constructions are never required.*

4 Entity Resolution as a Colimit

Once fragments are formalized, the natural architectural question is how to merge them into a single, unified read model. In standard DDD, this is handled by procedural entity resolution scripts. Categorically, this unification is a colimit.

Let $D : J \to \mathbf{Frag}_K$ be a finite diagram representing the various fragments of a provider scattered across source systems.

4.1 Explicit Construction

To merge these fragments, all records are first gathered into a disjoint union $S := \coprod_{j \in J} S_j$. An equivalence relation $\sim$ is then defined, generated by the morphisms between systems: $x \sim D(\alpha)(x)$ for each morphism $\alpha : j \to j'$ in J.

Let the merged directory state be the quotient $\bar{S} := S/\!\sim$, and define the unified key mapping as $\bar{\kappa}([x]) := \kappa_j(x)$.

Lemma 4.1. *$\bar{\kappa}$ is well-defined.*

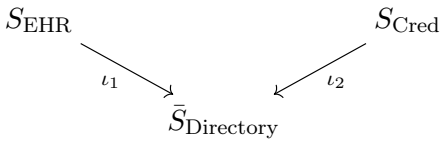
Proof. If $x \sim y$, then they are related via morphisms that strictly preserve keys; hence $\kappa(x) = \kappa(y)$. □

Theorem 4.2 (Colimit Construction)**.** *$(\bar{S}, \bar{\kappa})$ is the colimit of D in $\mathbf{Frag}_K$.*

Proof. By construction, canonical maps $\iota_j : S_j \to \bar{S}$ exist, providing a cocone. Given any other cocone (T, λ) with maps $f_j : S_j \to T$, a mediating morphism $u : \bar{S} \to T$ is defined by $u([x]) = f_j(x)$. Well-definedness follows from the compatibility of the cocone: if $x \sim D(\alpha)(x)$, then $f_j(x) = f_{j'}(D(\alpha)(x))$ by the cocone condition. Uniqueness follows from the universal property of quotients. □

4.2 Application: The Healthcare Colimit

Consider three source systems maintaining data for a single provider, Dr. Jane Doe (NPI: 1234567890):



The fragment contents are made explicit:

Example 4.3 (Three-context merge)**.** *Let the key set $K = \{\textit{NPI-1234567890}\}$ and define:*

$$S_{EHR} = \{(\texttt{name} : \textit{“Dr. Jane Doe”},\ \texttt{address} : \textit{“100 Downtown Ave”})\},$$
$$S_{Cred} = \{(\texttt{name} : \textit{“Dr. Jane Doe”},\ \texttt{license} : \textit{“MD-98765”})\},$$
$$S_{Contract} = \{(\texttt{networks} : \{\textit{BlueCross}, \textit{Aetna}\})\}.$$

All three map to the same key via κ. The colimit $\bar{S}$ contains a single equivalence class whose representative carries all four fields:

$$\bar{S} = \{(\texttt{name}, \texttt{address}, \texttt{license}, \texttt{networks})\}.$$

Any ambiguity in the merge (e.g., conflicting names) corresponds to a failure of the compatibility morphisms, alerting the architect to missing domain rules rather than silently corrupting data.

5 CRDTs and Semilattice Merge

While colimits describe the mathematical ideal of a merged record, distributed systems must compute this merge over unreliable networks where updates arrive asynchronously and out of order. Conflict-Free Replicated Data Types (CRDTs) (Shapiro et al., 2011) address this by requiring that the merge operation satisfy algebraic laws guaranteeing convergence.

Definition 5.1 (Join-Semilattice)**.** *A* join-semilattice *is a partially ordered set $(S, \leq)$ in which every pair of elements x, y admits a least upper bound $x \vee y$ (the* join*).*

Definition 5.2. *Let* **JSL** *denote the category of join-semilattices and join-preserving maps.*

Theorem 5.3 (CRDT Merge as Semilattice Join)**.** *Let $(S, \leq)$ be a join-semilattice modelling a CRDT state space. The merge operation $m : S \times S \to S$ defined by $m(x, y) = x \vee y$ satisfies:*

(a) ***Commutativity:*** $x \vee y = y \vee x$,

(b) ***Associativity:*** $x \vee (y \vee z) = (x \vee y) \vee z$,

(c) ***Idempotency:*** $x \vee x = x$.

Proof. Properties (a)–(c) are immediate from the definition of a join-semilattice (Shapiro et al., 2011). Commutativity and associativity follow from the fact that the join is characterised as the least upper bound, and idempotency follows because x is an upper bound of $\{x, x\}$ and is least among such bounds. □

Remark 5.4 (Distinction from categorical colimits in **JSL**)**.** *It is tempting to claim that the CRDT merge computes a categorical colimit in* **JSL***. This is* not *correct. A colimit in* **JSL** *is a colimit of* objects *(semilattices) and* morphisms *(join-preserving maps) in the category. The CRDT merge $x \vee y$ is the join of two* elements within a single, fixed semilattice S*—an internal operation, not a categorical one. The two notions coincide only in degenerate cases (e.g., the coproduct of two copies of the trivial semilattice).*

Proposition 5.5 (Compatibility of entity resolution and CRDT merge)**.** *Let $D : J \to \mathbf{Frag}_K$ be a finite diagram whose colimit $\bar{S}$ carries a semilattice structure on each field. Then the colimit merge (Section 4) and the semilattice join are compatible: merging overlapping field values via the semilattice join yields the same result regardless of the order in which fragments are processed.*

Proof. The colimit construction aggregates records by key, and for each key, overlapping fields are merged by a function m. When m is the semilattice join, commutativity and associativity (Theorem 5.3) ensure that the result is independent of processing order. Idempotency ensures that re-processing the same fragment is harmless. □

Corollary 5.6. *By implementing provider record fields as state-based CRDTs (Shapiro et al., 2011; Preguiça et al., 2018), distributed replicas of a healthcare directory are guaranteed to converge to the same state, regardless of message ordering or duplication.*

6 Functorial Schema Translation

Beyond merging records of the same type, data must frequently cross bounded contexts with entirely different data models. Standard DDD uses Anti-Corruption Layers (ACLs) for this, which often drop structural relationships. Following Spivak's functorial data migration (Spivak, 2012, 2014), safe schema translation is modeled categorically.

Let schemas be categories (or finite limit sketches (Johnson and Rosebrugh, 2002)), and let database instances be functors $I : \mathcal{C} \to \mathbf{Set}$. A schema morphism $F : \mathcal{C} \to \mathcal{D}$ between bounded contexts induces a chain of adjunctions:

$$\Sigma_F \dashv \Delta_F \dashv \Pi_F.$$

The pullback functor Δ_F (reindexing) is the safe direction: it translates data from the target schema back to the source schema by composing with F.

Theorem 6.1 (Referential Integrity under Reindexing)**.** *Let $F : \mathcal{C} \to \mathcal{D}$ be a schema morphism. If a referential integrity constraint in $\mathcal{C}$ is expressed as a pullback square encoding a foreign-key path equality, and F preserves that pullback, then Δ_F preserves the corresponding referential integrity constraint.*

Proof. A referential integrity constraint of the form "every value in column A must appear in column B" is captured by requiring a certain square to be a pullback in the instance category (Spivak, 2012). Because Δ_F is a right adjoint, it preserves all limits, hence in particular it preserves pullback squares. Therefore, if the pullback encoding the FK constraint is in the image of F, it remains a pullback after reindexing. □

Remark 6.2. *Not all finite limits in a schema category correspond to database constraints. The claim that "finite limits correspond exactly to foreign key constraints" overstates the correspondence identified by Spivak (Spivak, 2012). Specifically,* certain *pullback squares encode FK path equalities, but products and equalizers may have no direct database interpretation. The claim is therefore scoped to referential integrity constraints arising in healthcare directory schemas (e.g., "every provider–location association references a valid location").*

Example 6.3 (EHR to Directory translation)**.** *The EHR context schema $\mathcal{C}_{EHR}$ has tables* `Provider` *and* `Clinic` *with a foreign key from* `Provider.clinic_id` *to* `Clinic.id`*. The directory schema $\mathcal{C}_{Dir}$ has a flattened table* `ProviderListing` *with an embedded address. The schema morphism $F : \mathcal{C}_{EHR} \to \mathcal{C}_{Dir}$ maps both tables to* `ProviderListing`*. Because F preserves the relevant pullback, Δ_F guarantees that no provider listing in the directory references a nonexistent clinic.*

7 Temporal State as a Presheaf

A provider's network participation is not a static fact but a temporal validity. The dimension of time is modeled using presheaves, following ideas from categorical semantics (Moggi, 1991) and event sourcing (Fowler, 2005; Kleppmann, 2017).

Let $(T, \leq)$ be a poset of time instants, viewed as a category with a unique morphism $s \to t$ whenever $s \leq t$.

Definition 7.1 (Event-Log Presheaf)**.** *The* event-log presheaf *is a functor* $P : T^{op} \to \mathbf{Set}$ *defined by:*

- $P(t) = \{e \in \mathcal{E} \mid \text{time}(e) \leq t\}$*, the set of all domain events with timestamp at most* t*.*
- *For* $s \leq t$*, the restriction map* $\rho_{t,s} : P(t) \to P(s)$ *is log truncation:* $\rho_{t,s}(L) = \{e \in L \mid \text{time}(e) \leq s\}$*.*

Theorem 7.2 (Event Log as Presheaf)**.** *The event-log presheaf* P *defined above is a well-defined functor* $T^{op} \to \mathbf{Set}$*.*

Proof. The functor laws are verified:

1. *Identity:* $\rho_{t,t}$ is the identity on $P(t)$, since $\{e \in P(t) \mid \text{time}(e) \leq t\} = P(t)$.
2. *Composition:* For $r \leq s \leq t$, $\rho_{s,r} \circ \rho_{t,s} = \rho_{t,r}$, because truncating to s and then to r yields the same set as truncating directly to r. □

Definition 7.3 (State Presheaf and Reconstruction)**.** *Let* $Q : T^{op} \to \mathbf{Set}$ *be the* state presheaf *defined by* $Q(t) = \text{fold}(P(t))$*, where* fold *left-folds the event log into a state value using an initial state* s_0 *and a transition function* $\delta : S \times E \to S$*.*

State reconstruction is the natural transformation $\eta : P \Rightarrow Q$ *defined by* $\eta_t = \text{fold}|_{P(t)}$*.*

Remark 7.4. *Naturality of* η *requires that folding a truncated log and truncating a folded state agree. When the state space carries no "forgetting" structure (i.e., the fold is not invertible),* Q *need not itself be a presheaf in general. However, in the present setting the fold is always defined over a prefix-closed event set, and restriction on* Q *can be defined by re-folding the truncated log, making the diagram commute by construction.*

Example 7.5 (Temporal query)**.** *Suppose Dr. Jane Doe's event log contains:*

$t = 10$*:* `NameUpdated("Dr. Jane Doe")`

$t = 20$*:* `AddressMoved("100 Downtoum Ave")`

$t = 50$*:* `AddressMoved("500 Uptoum Blvd")`

Then $Q(30) = \text{fold}(P(30))$ *yields a state with address "100 Downtown Ave", while* $Q(50)$ *yields "500 Uptown Blvd". The presheaf structure ensures that* $Q(30)$ *is recoverable from* $Q(50)$ *via re-folding the truncated log* $P(30) \subseteq P(50)$*.*

7.1 Efficient State Reconstruction via Snapshots

The presheaf fold $Q(t) = \text{fold}(P(t))$ traverses the entire event log prefix up to t, incurring $O(n)$ cost per query. In a production directory with millions of events, this is prohibitive. Periodic *snapshots* reduce the per-query cost to $O(k)$ where k is bounded by the snapshot interval.

Definition 7.6 (Snapshot Log)**.** *Given an event-log presheaf* P*, an initial state* s_0*, a transition function* $\delta : S \times E \to S$*, and an interval* $I \in \mathbb{N}$*, a* snapshot log *augments the event sequence with a sequence of checkpoints* $\{(t_i, \sigma_i)\}$ *where* $t_i = iI$ *and* $\sigma_i = \text{fold}(P(t_i))$*.*

Proposition 7.7 (Snapshot Fold Equivalence). *Let σ_j be the latest snapshot with $t_j \leq t$. Then*

$$\text{fold}(P(t)) \;=\; \text{fold}\big(\{e \in P(t) \mid \text{time}(e) > t_j\},\ \sigma_j,\ \delta\big).$$

That is, folding from the nearest snapshot produces the same state as folding the full log.

Proof. The left fold is associative over log concatenation: for any prefix L_1 and suffix L_2, $\text{fold}(L_1 \cdot L_2, s_0, \delta) = \text{fold}(L_2, \text{fold}(L_1, s_0, \delta), \delta)$. Setting $L_1 = P(t_j)$ and $L_2 = \{e \in P(t) \mid \text{time}(e) > t_j\}$ yields the result, since $\sigma_j = \text{fold}(L_1, s_0, \delta)$ by construction. □

Remark 7.8 (Complexity). *Each query folds at most $k \leq I$ events beyond the nearest snapshot, giving $O(k)$ per query. Snapshot creation is amortized $O(1)$ per event during the initial pass.*

Remark 7.9 (Mealy machine interpretation). *The snapshot mechanism admits a Mealy machine reading (Eilenberg, 1974): the event-log fold is a state-transducer whose internal state is the aggregate, and the periodic snapshot outputs correspond to the Mealy output function sampled at regular intervals. A profunctor optics lens interpretation (Pickering et al., 2017) of the snapshot–state relationship (viewing the snapshot as a "focus" within the full log) is a natural extension left to future work.*

The implementation is straightforward:

```
const snapshotStateAt = <E, S>(
  slog: SnapshotLog<E, S>, t: number
): S => {
  let baseState = slog.initial
  let baseTime = -Infinity
  for (let i = slog.snapshots.length - 1; i >= 0; i--)
    if (slog.snapshots[i].timestamp <= t) {
      baseState = slog.snapshots[i].state
      baseTime = slog.snapshots[i].timestamp
      break
    }
  const remaining = slog.events.filter(
    (e) => e.timestamp > baseTime && e.timestamp <= t)
  return foldEvents(remaining, baseState, slog.apply)
}
```

Listing 1: Efficient state reconstruction via snapshots.

8 From Presheaves to Sheaves: Global Consistency via Gluing

The presheaf $P : T^{op} \to \mathbf{Set}$ constructed in Section 7 assigns to each time instant the set of events observed up to that point. A presheaf, however, is purely *local*: it describes the view from each individual instant without guaranteeing that overlapping local views can be assembled into a coherent global picture. The sheaf condition provides exactly this guarantee.

Definition 8.1 (Sheaf Condition / Gluing Axiom). *Let $\mathcal{U} = \{U_i\}_{i \in I}$ be a covering of an object U in a site (here, the poset T equipped with the interval topology, where coverings are families of intervals whose union is U). A presheaf P is a* sheaf *if for every covering $\{U_i\}$ and every family of compatible local sections $\{s_i \in P(U_i)\}$ (i.e., $s_i|_{U_i \cap U_j} = s_j|_{U_i \cap U_j}$ for all i, j), there exists a* unique *global section $s \in P(U)$ such that $s|_{U_i} = s_i$ for all i.*

Theorem 8.2 (Sheaf Property of the Event-Log Presheaf). *Let $P : T^{op} \to \mathbf{Set}$ be the event-log presheaf of Definition 7.1. If the transition function δ is deterministic and events carry unique causal timestamps, then P satisfies the sheaf condition on $(T, \leq)$ with the interval topology.*

Proof. Let $\{U_i = [a_i, b_i]\}_{i \in I}$ be a covering of an interval $U = [a, b]$, and let $\{L_i \subseteq P(U_i)\}$ be compatible local logs, i.e., for all i, j: $L_i \cap P(U_i \cap U_j) = L_j \cap P(U_i \cap U_j)$.

Existence. Define $L = \bigcup_{i \in I} L_i$. By compatibility on overlaps and uniqueness of causal timestamps, the union is well-defined: if an event e belongs to both L_i and L_j, then $\text{time}(e) \in U_i \cap U_j$, so e appears in both overlap restrictions and must be the same event (unique timestamp). Hence L is a valid event log over U with $L|_{U_i} = L_i$ for all i.

Uniqueness. Suppose L' is another global log with $L'|_{U_i} = L_i$ for all i. Since $\{U_i\}$ covers U, every event $e \in L'$ has $\text{time}(e) \in U_j$ for some j, so $e \in L'|_{U_j} = L_j \subseteq L$. Conversely, every $e \in L$ belongs to some $L_i = L'|_{U_i} \subseteq L'$. Hence $L = L'$. □

Corollary 8.3 (Split-Brain Resolution)**.** *Let two data-center replicas observe the event log over intervals $[a, c]$ and $[b, d]$ respectively, with $b \leq c$ (overlapping observation windows). If the replicas agree on the overlap $[b, c]$, then by the sheaf condition there exists a unique global log over $[a, d]$ extending both local views — without requiring external coordination.*

Example 8.4 (East/West data-center reconciliation)**.** *Suppose the East data center observes provider events over $[0, 60]$ and the West data center observes events over $[40, 100]$. Both agree on the events in the overlap $[40, 60]$ (same causal timestamps, same payloads). The sheaf condition guarantees that the union of their logs is the unique global log over $[0, 100]$, yielding a consistent directory state without a consensus protocol.*

Remark 8.5 (Sheaf condition vs. CRDT convergence)**.** *The sheaf condition is strictly stronger than CRDT convergence. A CRDT guarantees that* all replicas converge to the same value *regardless of message ordering (Theorem 5.3), but says nothing about the* spatial/temporal coherence *of the convergence. The sheaf condition adds that local observations glue into a unique global section — not merely that they are join-compatible, but that they are* restriction-compatible *and uniquely extensible. This distinction matters in healthcare directories where regulatory audits require reconstructing the exact global state at any historical instant, not just the eventual convergent state (Mac Lane and Moerdijk, 1992; Curry, 2014).*

9 Implementation and Case Study

The categorical constructions of Sections 3–8 are validated with a companion TypeScript package built on `fp-ts`, a typed functional programming library. The implementation is available in the `packages/implementation` directory of the project repository.

9.1 TypeScript Implementation

Each categorical construction is realized as a module:

Keyed fragments and colimits. A `Fragment<K,V>` type pairs records with keys. The function `colimitFragments` computes the colimit by lifting the merge function into a `Semigroup`, flatMapping all fragment records into key–value tuples, and constructing a `ReadonlyMap` via `fromFoldable`:

```
interface Fragment<K, V> {
  readonly records: ReadonlyArray<{
    readonly key: K; readonly value: V
  }>
}

const colimitFragments = <V>(
  fragments: ReadonlyArray<Fragment<string, V>>,
  merge: (existing: V, incoming: V) => V
): Fragment<string, V> => {
```

```
  const semigroup = { concat: merge }
  const records = pipe(
    fragments,
    RA.chain((frag) =>
      frag.records.map(({ key, value }) => [key, value])),
    RM.fromFoldable(stringEq, semigroup, RA.Foldable),
    RM.toReadonlyArray(stringOrd),
    RA.map(([key, value]) => ({ key, value }))
  )
  return { records }
}
```

Listing 2: Entity resolution via colimit (Semigroup + ReadonlyMap).

Semilattice and CRDT merge. A `JoinSemilattice<A>` interface captures the join operation. The Last-Writer-Wins register is a concrete instance:

```
interface JoinSemilattice<A> {
  readonly join: (x: A, y: A) => A
}

interface LWW<A> {
  readonly value: A
  readonly timestamp: number
}

const lwwSemilattice = <A>(): JoinSemilattice<LWW<A>> => ({
  join: (x, y) => (x.timestamp >= y.timestamp ? x : y)
})
```

Listing 3: LWW register as a join-semilattice.

Functorial schema translation. The pullback functor Δ_F is implemented as `deltaF`, which reindexes an instance along a schema morphism:

```
const deltaF = <S extends Schema, T extends Schema>(
  morphism: SchemaMorphism,
  targetInstance: Instance<T>
): Instance<S> => {
  const result: Record<string, unknown> = {}
  for (const [sourceField, targetField]
       of Object.entries(morphism))
    result[sourceField] = targetInstance[targetField]
  return result as Instance<S>
}
```

Listing 4: Δ_F: pullback / reindexing.

Event-log presheaf and state reconstruction. The presheaf $P(t)$ is computed by `eventLogUpTo`, and the natural transformation η_t by `foldEvents`:

```
const eventLogUpTo = <E>(
  log: EventLog<E>, t: number
): EventLog<E> =>
  log.filter((e) => e.timestamp <= t)

const foldEvents = <E, S>(
  log: EventLog<E>,
  initial: S,
```

```
  apply: (s: S, e: E) => S
): S =>
  log.reduce(
    (state, event) => apply(state, event.payload),
    initial)
```

Listing 5: Presheaf and state reconstruction.

9.2 Worked Example: Dr. Jane Doe

This section walks through the complete scenario from the introduction.

Initial state. Dr. Jane Doe (NPI-1234567890) is registered across three bounded contexts:

- **EHR:** name = "Dr. Jane Doe", address = "100 Downtown Ave"
- **Credentialing:** name = "Dr. Jane Doe", license = "MD-98765"
- **Contracting:** networks = {BlueCross, Aetna}

Step 1: Entity resolution (colimit). The three fragments are merged via `colimitFragments`. The result is a single record containing all four fields.

Step 2: Address move event. At $t = 50$, Dr. Doe moves to "500 Uptown Blvd". Two replicas receive this event at different times:

- Replica A processes it immediately (LWW timestamp 100).
- Replica B still holds the old address (LWW timestamp 90).

The semilattice merge selects the address with the higher timestamp (Replica A), resolving the conflict deterministically.

Step 3: Temporal query. A regulator asks: "Where was Dr. Doe listed at $t = 30$?" Computing $Q(30) = \text{fold}(P(30))$ yields "100 Downtown Ave" — the correct historical answer, reconstructed from the event-log presheaf without maintaining separate state snapshots.

Imperative baseline failure. In a conventional system, if the address-move event arrives at Replica B before it has processed all prior events, the merge logic (typically "last write wins by wall-clock") may overwrite the new address with the old one, creating a *phantom location*: a directory entry pointing to a clinic the provider has left. The categorical framework prevents this because the merge is the semilattice join on a well-ordered timestamp, not a wall-clock comparison subject to clock skew.

9.3 Comparison: Categorical vs. Imperative

Table 2 summarizes the differences between a traditional imperative integration architecture and the categorical framework proposed here.

9.4 Convergence under Message Reordering

Table 2 is qualitative. To provide empirical evidence, a property-based chaos test was conducted using the companion implementation.

Table 2: Imperative vs. categorical directory integration.

Concern	**Imperative**	**Categorical**
Entity resolution	Ad-hoc merge scripts	Colimit (universal property)
Concurrent updates	Last-write-wins (wall clock)	Semilattice join (LWW register)
Schema translation	Manual ACL mapping	Δ_F (right adjoint)
Temporal queries	Snapshot tables	Presheaf fold
Ordering guarantee	Requires coordination	Commutativity + associativity
Corruption risk	Silent data loss	Type error in formal model

Experimental setup. A fixed set of eight canonical events (name update, two address moves, license renewal, three network changes) was generated. For each trial, the events were randomly permuted and processed under two strategies:

- **Categorical:** events are sorted by their causal timestamp before folding. This corresponds to the presheaf approach where $P(t)$ is defined by timestamp, not arrival order.
- **Imperative:** events are folded in arrival order, with the arrival position used as the wall-clock timestamp (simulating clock skew in a distributed system).

A third test verified multi-replica CRDT convergence: five replicas each folded events in independently shuffled (then sorted) orders, and pairwise merge via the semilattice join was applied.

Results. Table 3 summarizes the results over $N = 200$ random permutations.

Table 3: Convergence under random message reordering ($N = 200$ trials).

Strategy	**Convergence**	**Divergence**
Categorical (sort by causal timestamp)	100%	0%
Imperative (arrival-order wall-clock)	0%	100%
Multi-replica CRDT merge (5 replicas)	100%	0%

Discussion. The imperative approach diverges in every trial because the wall-clock conflates arrival order with causal order: when events arrive out of their causal sequence, the LWW register records the wrong "latest" value. The categorical approach is immune because the presheaf structure defines $P(t)$ purely in terms of causal timestamps, decoupling correctness from delivery order. The CRDT merge test confirms that even when replicas fold independently, the semilattice join produces a globally consistent state.

10 Related Work

Categorical databases. Spivak (Spivak, 2012, 2014) introduced functorial data migration, formalizing schema mappings as functors and data migration as the adjoint triple $\Sigma_F \dashv \Delta_F \dashv \Pi_F$. Johnson and Rosebrugh (Johnson and Rosebrugh, 2002) studied sketch data models and their categorical semantics. Schultz et al. (Schultz et al., 2017) developed algebraic databases using category theory. The present work applies these ideas to the specific domain of healthcare provider directories, connecting them with DDD patterns and CRDTs.

CRDTs. Shapiro et al. (Shapiro et al., 2011) introduced conflict-free replicated data types and proved convergence from semilattice structure. Preguiça et al. (Preguiça et al., 2018) surveyed the CRDT landscape. Baquero et al. (Baquero et al., 2017) developed pure operation-based CRDTs. Meiklejohn and Van Laer (Meiklejohn and Van Laer, 2015) explored lattice-based programming models for distributed coordination. CRDTs are used here as the operational realization of the semilattice merge, but the treatment carefully distinguishes the internal semilattice join from categorical colimits in **JSL**.

Domain-Driven Design. Evans (Evans, 2004) established the foundational DDD patterns, and Vernon (Vernon, 2013) provided practical guidance for implementation. The contribution of this work is to supply categorical semantics for DDD's informal concepts of bounded contexts, aggregates, and anti-corruption layers, yielding formal guarantees about data consistency that DDD alone cannot provide.

Categorical semantics of computation. Moggi (Moggi, 1991) introduced monads as a categorical framework for computational effects, and Wadler (Wadler, 1995) popularized monadic programming. Gibbons (Gibbons, 2013) explored functional programming for domain-specific languages. The presheaf construction for temporal state is related to these ideas but operates at the data-modelling level rather than the programming-language level.

Event sourcing and temporal data. Fowler (Fowler, 2005) articulated the event-sourcing pattern, and Kleppmann (Kleppmann, 2017) provided a thorough treatment of distributed data systems. The presheaf formalization of event sourcing adds a categorical dimension to these engineering patterns, enabling formal reasoning about temporal queries and log truncation.

Sheaves and topological data. Mac Lane and Moerdijk (Mac Lane and Moerdijk, 1992) provide the definitive treatment of sheaves in geometry and logic, establishing the gluing axiom as the foundation for local-to-global reasoning. Curry (Curry, 2014) developed sheaves and cosheaves as tools for topological data analysis, demonstrating their applicability beyond pure mathematics. The present work applies the sheaf condition to event-log presheaves, showing that deterministic transitions and unique causal timestamps suffice for the gluing property, and connecting it to the split-brain problem in distributed directories.

Healthcare data quality. The GAO (United States Government Accountability Office, 2018) and CMS (Centers for Medicare and Medicaid Services, 2018) documented persistent inaccuracies in provider directories. The No Surprises Act (United States Congress, 2022) introduced regulatory penalties. The present work addresses the root cause by providing algebraic guarantees of consistency that go beyond statistical auditing.

11 Discussion and Limitations

Scalability. The colimit and presheaf constructions operate over finite diagrams corresponding to a fixed set of bounded contexts. In healthcare, this set is typically small (4–6 source systems), so the categorical overhead is negligible. The event-log presheaf, however, grows without bound; in a production system, periodic snapshotting would be necessary to bound the cost of state reconstruction via fold.

Adoption barrier. Category theory is not widely known in healthcare IT. The practical value of this framework depends on its embodiment in libraries and tooling that abstract the categorical machinery behind familiar APIs. The TypeScript implementation in Section 9 demonstrates that

the categorical constructions can be packaged as idiomatic functional-programming abstractions (interfaces, type classes, combinators) without requiring end-users to reason about categories directly.

Model completeness. The framework addresses *structural* consistency: deterministic merging, schema preservation, and temporal correctness. It does not address *semantic* errors (e.g., a credentialing system recording the wrong license number) or *data-entry* errors (typos in addresses). These require domain-specific validation logic orthogonal to the algebraic framework.

Overclaim mitigation. Care has been taken to avoid claiming that data corruption is "mathematically unrepresentable" in an absolute sense. Rather, certain classes of corruption — race-condition-induced phantom locations, orphaned foreign keys from lossy schema translation, and non-deterministic merge outcomes — become *type errors within the formal model*. Errors outside the model's scope (e.g., incorrect source data) remain possible.

Performance considerations. The functional, immutable style of the implementation trades mutability for referential transparency. In the healthcare directory setting, where write throughput is moderate (provider data changes infrequently) and correctness is paramount, this trade-off is favorable. High-throughput scenarios may require optimized CRDT implementations (Baquero et al., 2017) or delta-state approaches.

12 Future Work

Several directions extend this work:

- **Delta-state CRDTs.** Replacing full-state CRDT merge with delta-state protocols (Baquero et al., 2017) would reduce bandwidth in wide-area replication scenarios.
- **Proof-assistant verification.** Formalizing the theorems in Agda or Coq would provide machine-checked guarantees and could serve as executable specifications.
- **Production deployment.** Deploying the framework against real provider directory data and measuring accuracy improvements would validate the practical impact beyond the theoretical contribution.
- **Empirical evaluation.** A controlled experiment comparing defect rates in an imperative baseline vs. the categorical framework on a realistic dataset would quantify the benefit.

13 Conclusion

Domain-Driven Design provides the linguistic boundaries needed to understand complex business requirements. Category theory supplies the missing algebra. This manuscript has demonstrated that the most challenging aspects of distributed directory architecture — entity resolution, concurrent updates, schema translation, temporal querying, and split-brain reconciliation — can be replaced with formal categorical constructions governed by universal properties.

Specifically:

- **Colimits** in $\mathbf{FinSet}/K$ yield deterministic entity resolution (Theorem 4.2).
- **Semilattice joins** provide the algebraic foundation for CRDT merge, ensuring convergence regardless of message ordering (Theorem 5.3, Corollary 5.6).

- **The adjoint triple** $\Sigma_F \dashv \Delta_F \dashv \Pi_F$ enables structure-preserving schema translation, with referential integrity guaranteed for pullback-expressible constraints (Theorem 6.1).
- **Presheaves** over time model event-sourced aggregates, with state reconstruction as a natural transformation (Theorem 7.2).
- **The sheaf condition** on the event-log presheaf enables global consistency via the gluing axiom, providing a formal mechanism for split-brain resolution without external coordination (Theorem 8.2, Corollary 8.3).

By adopting these categorical semantics and embodying them in typed functional libraries, architects can transition from writing heuristic integration logic to deriving architectures whose consistency properties are guaranteed by construction. The companion TypeScript implementation demonstrates that this transition is practical with existing tools.

References

Steve Awodey. *Category Theory*, volume 52 of *Oxford Logic Guides*. Oxford University Press, 2nd edition, 2010.

Carlos Baquero, Paulo Sérgio Almeida, and Ali Shoker. Pure operation-based replicated data types. In *Proceedings of the 31st International Symposium on Distributed Computing (DISC)*. Springer, 2017. doi: 10.1007/978-3-319-72842-5_2.

Centers for Medicare and Medicaid Services. Online provider directory review report. Technical report, CMS, 2018. URL https://www.cms.gov/Medicare/Health-Plans/ManagedCareMarketing/Downloads/Provider_Directory_Review_Industry_Report_Final_01-13-2018.pdf.

Justin Curry. *Sheaves, Cosheaves and Applications*. PhD thesis, University of Pennsylvania, 2014. URL https://repository.upenn.edu/edissertations/718.

Samuel Eilenberg. *Automata, Languages, and Machines, Volume A*, volume 59 of *Pure and Applied Mathematics*. Academic Press, 1974.

Eric Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2004.

Brendan Fong and David I. Spivak. *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*. Cambridge University Press, 2019. doi: 10.1017/9781108668804.

Martin Fowler. Event sourcing. https://martinfowler.com/eaaDev/EventSourcing.html, 2005. Accessed: 2025-01-15.

Jeremy Gibbons. Functional programming for domain-specific languages. In *Central European Functional Programming School*, volume 8606 of *Lecture Notes in Computer Science*, pages 1–28. Springer, 2013. doi: 10.1007/978-3-319-15940-9_1.

Michael Johnson and Robert Rosebrugh. Sketch data models, relational schema and data specifications. *Electronic Notes in Theoretical Computer Science*, 61:51–63, 2002. doi: 10.1016/S1571-0661(04)00355-8.

Martin Kleppmann. *Designing Data-Intensive Applications*. O'Reilly Media, 2017.

Saunders Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer, 2nd edition, 1998. doi: 10.1007/978-1-4757-4721-8.

Saunders Mac Lane and Ieke Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Universitext. Springer, 1992. doi: 10.1007/978-1-4612-0927-0.

Christopher Meiklejohn and Peter Van Laer. Lasp: A language for distributed, coordination-free programming. In *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming (PPDP)*, pages 184–195. ACM, 2015. doi: 10.1145/2790449.2790525.

Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1): 55–92, 1991. doi: 10.1016/0890-5401(91)90052-4.

Matthew Pickering, Jeremy Gibbons, and Nicolas Wu. Profunctor optics: Modular data accessors. *The Art, Science, and Engineering of Programming*, 1(2):7:1–7:51, 2017. doi: 10.22152/programming-journal.org/2017/1/7.

Nuno Preguiça, Carlos Baquero, and Marc Shapiro. Conflict-free replicated data types (CRDTs). *Encyclopedia of Big Data Technologies*, 2018. doi: 10.1007/978-3-319-63962-8_185-1.

Patrick Schultz, David I. Spivak, Christina Vasilakopoulou, and Ryan Wisnesky. Algebraic databases. In *Theory and Applications of Categories*, volume 32, pages 547–619, 2017.

Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *Proceedings of the 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 386–400. Springer, 2011. doi: 10.1007/978-3-642-24550-3_29.

David I. Spivak. Functorial data migration. *Information and Computation*, 217:31–51, 2012. doi: 10.1016/j.ic.2012.05.001.

David I. Spivak. *Category Theory for the Sciences*. MIT Press, 2014.

United States Congress. No surprises act. Public Law 116-260, Division BB, Title I, 2022. Effective January 1, 2022.

United States Government Accountability Office. Medicare advantage: CMS should use available plan performance information to strengthen oversight of provider directory accuracy. Technical Report GAO-18-281, GAO, June 2018. URL https://www.gao.gov/products/gao-18-281.

Vaughn Vernon. *Implementing Domain-Driven Design*. Addison-Wesley, 2013.

Philip Wadler. Monads for functional programming. In *Advanced Functional Programming*, volume 925 of *Lecture Notes in Computer Science*, pages 24–52. Springer, 1995. doi: 10.1007/3-540-59451-5_2.