DEFENSIVE PUBLICATION — TECHNICAL DISCLOSURE (TDCOMMONS DEPOSIT COPY) · 2026-06-25

Keywords: defensive-publication, prior-art, ai-agents, messaging, multi-channel, jsonb, identity-resolution

Unified Multi-Channel Messaging History

A Single-Table, Polymorphic-Metadata Design for Cross-Channel Agent Conversation Continuity

Technical Defensive Publication — Public Prior Art

Field	Value
Title	Unified Multi-Channel Messaging History
Subtitle	A single-table, polymorphic-metadata design for cross-channel agent conversation continuity
Document type	Technical Defensive Publication (public prior art)
Publisher / copyright holder	Gus IT LLC (Florida, USA)
Author	Gustavo Assuncao, PhD
Publication date	2026-06-25
Version	1.0
Classification	Public
License	AGPL-3.0-or-later (copyleft; commercial license available)
Deposit channel	To be assigned (IP.com / Zenodo / arXiv) — no DOI asserted

Abstract

An AI agent that converses with the same person across several messaging providers — WhatsApp, Telegram, Signal, Discord — conventionally keeps one message history *per provider*. When a contact moves between channels the agent loses continuity: the earlier exchange is in another store, behind another code path, keyed by another identity. The two common remedies both fail: a table-per-provider design forces a schema migration every time a new channel is onboarded, and a lowest-common-denominator schema discards the provider-specific fields (Discord guild id, WhatsApp template id, Telegram thread id) that the agent later needs.

This publication discloses a design that resolves both failures simultaneously. **(1)** Every interaction across every provider is stored in **one relational table** whose typed columns are exactly the cross-channel invariants — `channel`, `contact`, `direction`, `body`, `occurred_at` — plus **one polymorphic metadata column** (JSONB) that absorbs each provider's idiosyncratic fields **with no schema migration when a new provider is added**. **(2)** A **contact-anchored timeline view** resolves a contact's several channel-specific identities (E.164 phone, @handle, numeric user-id, email) to a **canonical contact** and returns the time-ordered union of that contact's interactions across all channels as a single thread. **(3)** That single timeline — not a per-channel slice — is what the agent reads as conversation memory.

The contribution is the *specific combination*: migration-free polymorphic metadata, an identity-resolution join that anchors the timeline on a **canonical contact rather than on a channel**, and an agent-context

framing in which the unified timeline is the model's working memory. The design is published defensively to keep it free to practice. This document is exhaustive and enabling: it gives architecture, mechanics, the data model, a runnable clean-room reference, a worked example, failure-mode analysis, framework mapping, an evaluation methodology, and a full set of disclosed claims.

---

1. Executive Summary

1.1 Thesis

Cross-channel conversation continuity for an AI agent is best achieved not by a federation of per-channel stores stitched together at read time, and not by a brittle table-per-provider schema, but by a **single normalized table of cross-channel invariants plus a polymorphic metadata column**, read through a **contact-anchored timeline view** that joins on a *canonical contact identity* derived from channel-specific identifiers. This makes onboarding a new channel a zero-migration operation and makes "the conversation" a property of the **contact**, not of the **channel**.

1.2 Contributions

#	Contribution	Where
C1	Single-table schema of cross-channel invariant columns	§6, §8
C2	Polymorphic `metadata` column (JSONB) that absorbs provider-specific fields with **no migration** on new-provider onboarding	§7.2, §8
C3	Canonical-contact identity resolution from heterogeneous channel identifiers (phone / handle / user-id / email)	§7.3
C4	Contact-anchored timeline view: union across channels, ordered by time, one thread per contact	§7.4, §10
C5	Agent-context framing: the unified timeline is the model's conversation memory, not a per-channel slice	§7.5
C6	Append-only, idempotent ingest keyed by `(channel, provider_message_id)` for at-least-once provider webhooks	§7.6, §11
C7	Clean-room reference implementation reducing the design to practice	§9, Appendix C

1.3 Headline claim

> A method for unified storage of agent–contact messaging interactions across a plurality of messaging providers, comprising storing each interaction in a single table having a channel-identifier column, a contact-identifier column, a direction column, and a metadata column of polymorphic structured type, and surfacing per-contact timelines by joining across said channel-identifier column on a **canonical** contact identity resolved from the contact-identifier column.

1.4 Scope — what this is / is NOT

It is: a storage-and-read design for cross-channel agent message history; a migration-free schema pattern; a contact-identity resolution + timeline-join method; a clean-room reference and deployment plan.

It is NOT: a messaging transport or gateway (it sits behind whatever delivers provider messages); a CRM product; a claim to have invented unified inboxes; an assertion of certified compliance with any standard (we claim *semantic alignment* only, §12); a vendor SDK. The reference code is illustrative, not production, and contains no proprietary material.

---

2. Introduction & Motivation

2.1 The concrete problem

Consider an autonomous support/sales agent reachable on four providers. A customer, Dana, asks about an invoice on **WhatsApp** in the morning, gets no immediate answer, and pings the same business on **Telegram** in the afternoon: "any update on what I asked earlier?" If the agent stores history per channel, the Telegram worker queries the Telegram history for Dana's @handle, finds nothing relevant (the invoice question was on WhatsApp under a phone number), and answers as if the conversation is brand new. Dana experiences an agent with amnesia. This is the **channel-fragmentation tax**.

2.2 The tax, quantified qualitatively

The tax has several line items:

- **Continuity loss** — context that exists in store A is invisible to a worker reading store B; the agent re-asks, contradicts itself, or mishandles a handoff.
- **Migration tax** — a table-per-provider design requires a DDL migration (new table, new code path, new query) every time a fifth or sixth channel is added. Each migration is a deploy, a review, a rollback risk.
- **Field-loss tax** — a lowest-common-denominator unified schema throws away provider-specific fields the agent later needs (a WhatsApp template id for a compliant business reply; a Discord guild/channel id to reply in-thread; a Telegram `message_thread_id` for topic threads). Re-adding such a field is *also* a migration.
- **Identity tax** — the same human appears as `+15551234567` on WhatsApp, `@dana` on Telegram, `dana#0007` / a numeric id on Discord, and an email on a fallback channel. Without resolution, these are four "contacts."

2.3 Why existing approaches fall short

- **Per-channel stores** minimise migration friction per channel but maximise fragmentation — the read side has to fan out and reconcile, and identity is unresolved.
- **Table-per-provider** is clean per provider but pays the migration tax on every onboarding and still leaves identity unresolved.
- **LCD unified schema** unifies reads but pays the field-loss tax and re-migrates to add any provider-specific field.
- **Generic CRM unified inboxes** (HubSpot, Intercom, Front, Zendesk) solve the *human-agent inbox* problem but are not framed as an **AI agent's working memory**, do not expose a single time-ordered per-contact thread as the model's context, and are closed products rather than a migration-free schema pattern an AI platform can adopt internally.

2.4 Why now

Autonomous agents now routinely operate across messaging gateways (WhatsApp Business, Telegram Bot API, Signal, Discord). The unit of memory for such an agent should be the **contact**, not the **channel**, because the human thinks in terms of "my conversation with this business," not "my WhatsApp conversation" versus "my Telegram conversation." The design below makes the contact the unit of memory at the storage layer, cheaply and without migration churn.

---

3. Problem Statement

3.1 Formal framing

Let $P = \{p_1..p_m\}$ be messaging providers. Each provider p_i delivers interactions x carrying: a channel tag $c(x)=p_i$, a direction $d(x) \in \{inbound, outbound\}$, a human-readable body $b(x)$, an occurrence time $t(x)$, a provider-native contact identifier $id_i(x)$ (whose *form* varies by provider), and a provider-native bag of extra fields $meta(x)$ (whose *shape* varies by provider).

We want a store S and a read function $timeline(k)$ such that:

1. **Uniform write:** any x from any p_i is stored by one $append(x)$ with no provider-specific table or DDL.
2. **No-migration extensibility:** adding p_{m+1} (new channel, new $meta$ shape) requires **no** change to S's schema.
3. **Canonical identity:** a resolver κ maps heterogeneous $id_i(x)$ to a canonical contact key $k = \kappa(id_i(x))$, so the same human across providers maps to one k.
4. **Contact-anchored read:** $timeline(k)$ returns $\{ x \in S : \kappa(contactId(x)) = k \}$ ordered by $t(x)$, across all channels, as one thread.
5. **Lossless metadata:** $meta(x)$ is preserved verbatim and queryable.

3.2 Derived requirements

Req	Requirement	Satisfied in
R1	One table stores interactions from all providers	§6, §8
R2	Cross-channel invariants are typed columns	§6, §8
R3	Provider-specific fields are stored in one polymorphic column	§7.2, §8
R4	Onboarding a new provider requires **no** schema migration	§7.2, §12, §13
R5	Heterogeneous channel identifiers resolve to a canonical contact	§7.3
R6	A timeline view joins across channels by canonical contact, time-ordered	§7.4, §10
R7	The unified timeline is the agent's conversation memory	§7.5, §10
R8	Ingest is idempotent under at-least-once provider webhooks	§7.6, §11
R9	Metadata is queryable (filter/index) without flattening to columns	§7.2, §8.4
R10	Store is append-only / auditable	§8.3, §11
R11	Configuration (channels, id-normalisation rules) is externalised	§7.7, §9.3
R12	Failure modes degrade safely (partial provider outage, bad metadata)	§11

4. Related Work & Prior Art

This design is built deliberately *on* well-established practice; the novelty is a narrow combination. We name real prior sources and our adoption rationale.

- **Unified-inbox CRM / helpdesk suites — HubSpot, Intercom, Front, Zendesk, Salesforce Service Cloud.** These unify customer conversations from email, chat, and social into one timeline for a *human* agent. **Adopted idea:** the unified per-contact conversation timeline. **Not adopted / differs:** they are closed products, not a migration-free schema pattern; identity unification is a product feature, not a published join; and none frame the timeline as an *AI agent's working memory* fed to a model.
- **Polymorphic / EAV and document-in-relational patterns — PostgreSQL `JSONB`, MongoDB documents, the Entity-Attribute-Value pattern, "single-table design" in Amazon DynamoDB. Adopted idea:** store variant fields in one flexible column rather than migrating schema per variant. **Differs:** these are generic storage patterns; the contribution here is applying a typed-invariants + JSONB-variant split *specifically* to cross-provider messaging plus the contact-anchored read.
- **Customer Data Platforms & identity resolution — Segment, RudderStack, mParticle; deterministic identity stitching. Adopted idea:** resolve multiple external identifiers to one canonical profile. **Differs:** CDPs resolve identity for analytics/marketing, not to assemble a single agent-readable message thread, and run as separate heavyweight pipelines.
- **Messaging gateway abstractions — Twilio Conversations, MessageBird/Bird, Sendbird, the Matrix protocol's bridges. Adopted idea:** a normalized envelope over heterogeneous providers. **Differs:** these normalize *transport* and conversations as a service; they do not publish the *storage* design of a single JSONB-metadata table read as agent memory, and bridges replicate rooms rather than collapse to one contact timeline.

- **Chat/agent memory stores — LangChain/LlamaIndex conversation memory, vector chat-history stores. Adopted idea:** a model reads prior turns as memory. **Differs:** these are typically single-channel/session memories; they do not address cross-provider identity resolution or migration-free provider extension.
- **Event sourcing / append-only logs (Kafka, the log as system of record). Adopted idea:** append-only, idempotent ingest with a stable dedupe key. **Differs:** event logs are general infrastructure; here the append-only message table is the directly-queried system of record for a contact timeline.

We build on all of the above. The published delta is in §5.

---

5. Prior-Art Delta

Per novel feature: what the closest prior source has, what it lacks, and what this disclosure adds.

Prior source	What it has	What it LACKS	What THIS adds
Unified-inbox CRM (HubSpot/Intercom/Front)	Per-contact unified conversation timeline across channels (human-agent inbox)	Migration-free polymorphic provider-metadata column; open schema pattern; framing as an AI agent's model-fed memory	A published single-table + JSONB schema whose per-contact timeline is the **agent's** conversation memory
JSONB / single-table / EAV patterns	One flexible column for variant fields; no per-variant migration	Application to cross-provider messaging; canonical-contact timeline join	The specific invariants-columns + JSONB-variant split for messaging, **plus** the contact-anchored read view
CDP identity resolution (Segment/mParticle)	Deterministic stitching of many identifiers → one profile	A single time-ordered cross-channel **message thread** assembled from that identity; lightweight in-store join	A timeline view that uses canonical contact identity to **assemble one agent-readable thread**
Messaging gateways (Twilio Conversations, Matrix bridges)	Normalized cross-provider envelope; conversations-as-a-service	Disclosed **storage** schema with migration-free metadata; collapse to one contact timeline (vs replicated rooms)	A storage design where one JSONB-metadata table + contact join *is* the cross-channel history
Table-per-provider message stores	Clean provider-specific schemas	Migration-free onboarding; unified read; identity resolution	One table for all providers — **zero DDL** to add a provider — read as one contact timeline
Agent/chat memory libraries	Model reads conversation turns as memory	Cross-provider identity; provider-metadata preservation; no-migration extension	Cross-channel, identity-resolved, metadata-preserving memory for the agent

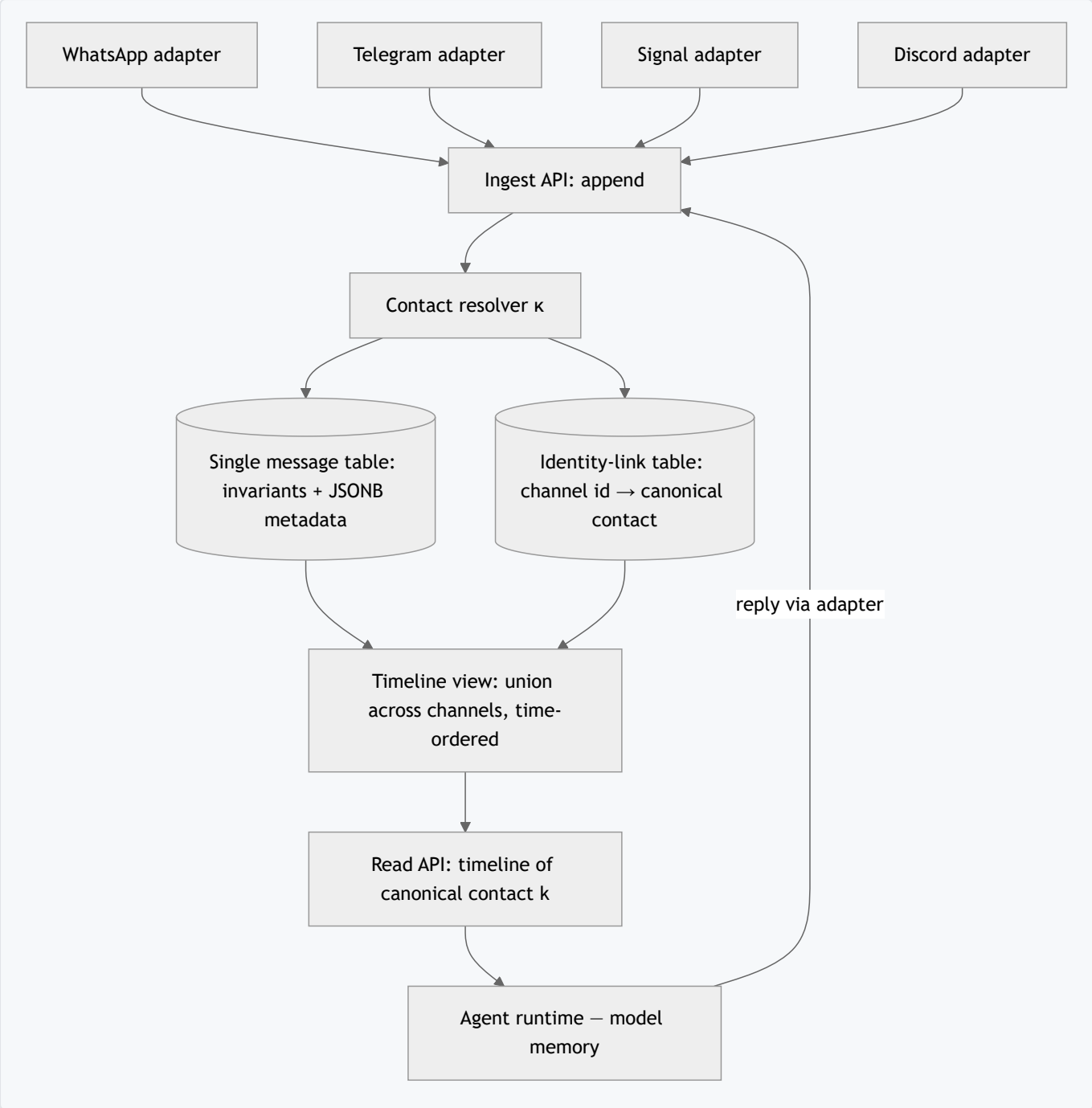
---

6. System Architecture

6.1 Components

- **Provider adapters (out of scope, upstream):** whatever delivers/sends provider messages. They hand normalized interactions to the ingest API. This design does not reimplement transports.
- **Ingest API (`append`):** validates the cross-channel invariants, places provider-specific fields into `metadata`, computes/looks up the canonical contact key, and performs an **idempotent** insert keyed by `(channel, provider_message_id)`.
- **Message store (single table):** the append-only table of interactions (§8). Typed invariant columns + one JSONB `metadata` column.
- **Contact resolver (κ):** maps a channel-native identifier (E.164 phone, @handle, numeric user-id, email) to a canonical contact id, using configurable normalisation + an identity-link table.
- **Timeline view / read API (`timeline`):** returns the time-ordered union of a canonical contact's interactions across all channels.
- **Agent runtime (consumer):** calls `timeline(k)` and feeds the resulting thread to the model as conversation memory.

6.2 Architecture diagram



6.3 Cross-cutting properties

- **Migration-free extensibility (R4):** new providers add rows, never columns.
- **Append-only (R10):** writes are inserts; corrections are new rows, not updates.
- **Idempotent ingest (R8):** at-least-once webhooks are de-duplicated by a stable key.
- **Configuration-driven (R11):** the channel set and per-channel identity normalisation rules are external configuration, not hard-coded.
- **Contact-centric (R5–R7):** the join key is a canonical contact, so "the conversation" survives a channel switch.

---

7. Detailed Mechanics

7.1 Cross-channel invariant columns

The fields common to *every* messaging provider are promoted to typed columns: `channel` (provider tag), `contact_raw` (the provider-native identifier as received), `contact_id` (the resolved canonical contact key), `direction` (`inbound`/`outbound`), `body` (human-readable text, nullable for media-only), `occurred_at` (provider event time), `received_at` (store time), `provider_message_id` (provider's id for this message, for idempotency). These are queryable and indexable directly.

7.2 Polymorphic metadata column (R3, R4, R9)

Everything *not* common across providers goes into one `metadata` JSONB column: WhatsApp `template_id` / `wa_business_id`; Telegram `chat_id`, `message_thread_id`; Discord `guild_id`, `channel_id`, `nick`; Signal `group_id`; delivery receipts; attachment descriptors; reply-to ids. Because JSONB is schemaless at the column level, **onboarding a new provider with a new metadata shape requires no DDL** — the new shape is simply a new JSON document in the same column. Metadata stays queryable via JSON path operators and can be indexed (GIN, or expression indexes on hot keys) without flattening to columns (R9).

```
metadata examples (illustrative):
  whatsapp : { "template_id":"invoice_v2", "wa_business_id":"…", "delivery":"read" }
  telegram : { "chat_id":99001, "message_thread_id":7, "is_topic":true }
  discord  : { "guild_id":"…","channel_id":"…","reply_to":"…" }
  signal   : { "group_id":"…", "expires_in":86400 }
```

7.3 Canonical contact resolution κ (R5)

A contact appears under different *forms* per channel. The resolver normalises and links them:

1. **Normalise** the provider-native identifier by channel rule (configurable): phone → E.164; Telegram/Discord → lower-cased `@handle` or numeric user-id; email → lower-cased, trimmed.
2. **Look up** the normalised identifier in an **identity-link table** (`(channel, identifier) → contact_id`).
3. **On hit**, use the stored canonical `contact_id`. **On miss**, mint a new canonical `contact_id` and insert the link (so future messages on this channel resolve to it). Operators may later *merge* two canonical ids when they discover they are the same human (e.g. a phone and a handle verified to belong to Dana); merging re-points links and is auditable.

The resolver is deliberately **deterministic and explicit** (an identity-link table), not a probabilistic stitch — this keeps the agent's memory boundaries auditable. Probabilistic stitching can feed *proposed* links for human/policy confirmation but never silently merges.

7.4 Contact-anchored timeline (R6)

`timeline(contact_id)` selects all rows whose resolved `contact_id` matches, across **all** channels, ordered by `occurred_at`, optionally windowed/paginated. The result is one thread that interleaves WhatsApp, Telegram, Signal, and Discord turns in true time order. Crucially the join key is the **canonical contact**, not the channel — so a channel switch is invisible to the read side.

7.5 Agent-context framing (R7)

The agent runtime, before replying on *any* channel, calls `timeline(contact_id)` and supplies the recent slice as the model's conversation memory. The model sees "the conversation with Dana" — including the morning WhatsApp invoice question — when replying on Telegram in the afternoon. The reply is then appended (with the reply channel's metadata) through the same `append`, closing the loop.

7.6 Idempotent append (R8)

Provider webhooks are at-least-once: the same message can arrive twice. `append` performs an upsert keyed by `(channel, provider_message_id)` — a duplicate is a no-op (or refreshes mutable delivery metadata only). The store therefore never double-counts a message, even under retries or replays.

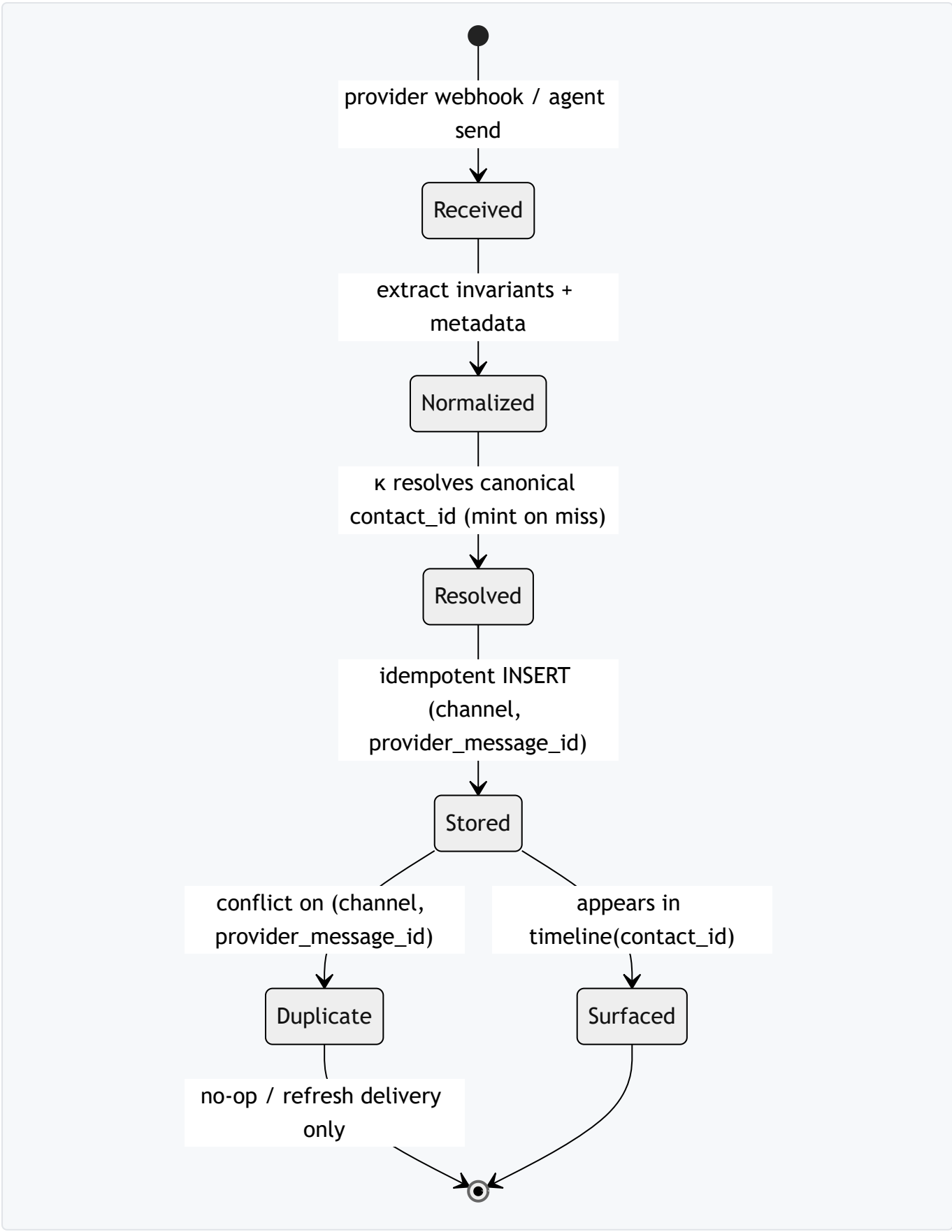
7.7 Control flow & configuration

```
append(interaction):
  inv  ← extract invariants(channel, contact_raw, direction, body, occurred_at,
                            provider_message_id)
  meta ← interaction \ inv            # everything else → JSONB
  cid  ← resolve(channel, normalize(channel, contact_raw))   # κ, mints on miss
  INSERT row(inv, contact_id=cid, metadata=meta)
        ON CONFLICT (channel, provider_message_id) DO NOTHING/refresh-delivery
  return cid

timeline(contact_id, window):
  return SELECT * FROM messages
         WHERE contact_id = contact_id
         ORDER BY occurred_at ASC
         [LIMIT/OFFSET or keyset by window]
```

Configuration points (R11): the **channel set**; per-channel **normalisation rules**; **metadata-key allowlist/index hints**; **timeline window size** and pagination strategy; whether duplicate refresh updates delivery metadata.

7.8 State machine of an interaction



7.9 Error handling

- **Malformed metadata:** stored as-is in JSONB if it is valid JSON; if not, the raw payload is wrapped (`{"_raw": "…"}`) so nothing is lost (R12).
- **Missing `provider_message_id`:** a deterministic surrogate is derived from `(channel, contact, occurred_at, hash(body))` so idempotency still holds.
- **Resolver miss:** mints a new contact (never drops the message).
- **Partial provider outage:** unaffected channels keep writing; the timeline just has a gap for the down channel — reads never fail because one provider is down.

---

8. Data Model

8.1 Tables

`messages` (the single store) and `identity_links` (canonical-contact resolution).

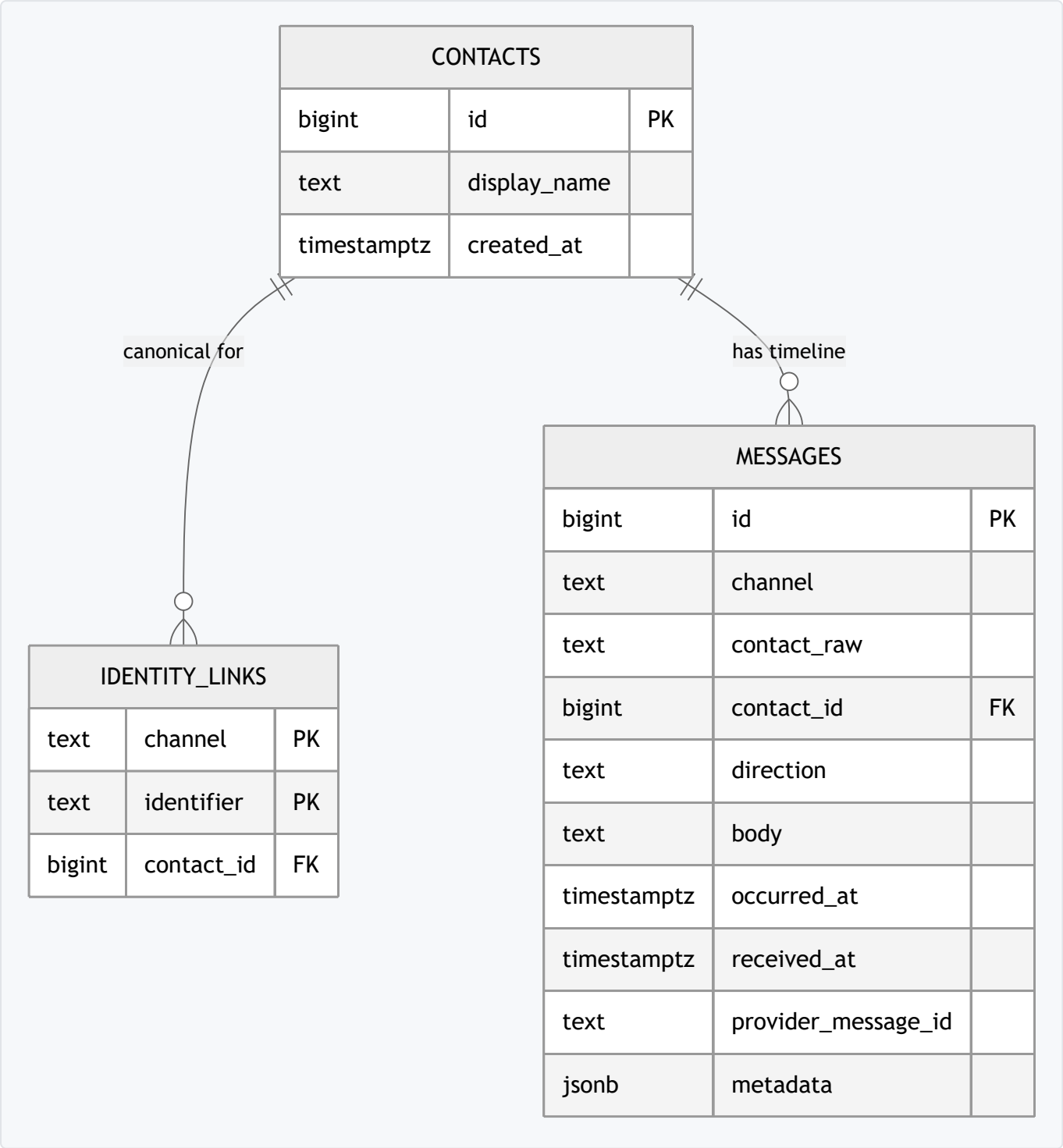
`messages` column	Type	Notes
`id`	bigserial PK	surrogate
`channel`	text NOT NULL	provider tag (whatsapp/telegram/signal/discord/…)
`contact_raw`	text NOT NULL	provider-native identifier as received
`contact_id`	bigint NOT NULL	canonical contact key (FK → contacts)
`direction`	text NOT NULL	`inbound` / `outbound`
`body`	text NULL	message text (null for media-only)
`occurred_at`	timestamptz NOT NULL	provider event time
`received_at`	timestamptz NOT NULL DEFAULT now()	store time
`provider_message_id`	text NOT NULL	provider's message id (idempotency)
`metadata`	jsonb NOT NULL DEFAULT '{}'	**polymorphic** provider-specific fields

Indexes: unique `(channel, provider_message_id)` for idempotency; btree `(contact_id, occurred_at)` for timeline reads; GIN on `metadata` for JSON filters.

`identity_links` column	Type	Notes
`channel`	text NOT NULL	provider tag
`identifier`	text NOT NULL	normalised channel-native identifier
`contact_id`	bigint NOT NULL	canonical contact key
PK	`(channel, identifier)`	one link per channel-identifier

A small `contacts` table (`id`, `display_name`, `created_at`) anchors canonical ids; merges re-point `identity_links.contact_id`.

8.2 ER diagram



8.3 Append-only posture (R10)

`messages` is written by INSERT only. Corrections (edits, deletes-by-sender, delivery updates) are represented either as new rows referencing the original via `metadata.replaces` / `metadata.reply_to`, or as a bounded refresh of *delivery* metadata on the existing row — content is never destructively overwritten, preserving an audit trail.

8.4 Metadata queryability (R9)

JSONB allows `metadata->>'template_id' = 'invoice_v2'` style filters and GIN / expression indexes on hot keys, so provider-specific fields are first-class for querying *without* being promoted to columns — preserving migration-free extensibility.

---

9. Reference Implementation & Enablement

9.1 What `src/` demonstrates

`src/` contains an original, dependency-light clean-room implementation:

- **`schema.sql`** — the `contacts`, `identity_links`, `messages` DDL with the idempotency unique index and the timeline index (PostgreSQL-flavoured, also runnable as the conceptual model in the in-memory harness).
- **`unified-history.js`** — an in-memory store implementing `append()`, canonical-contact resolution `resolve()`, idempotent insert, and `timeline(contactId)`, with externalised channel/normalisation configuration.
- **`demo.js`** — a self-checking example: Dana asks on WhatsApp, follows up on Telegram; the demo asserts both turns appear in **one** time-ordered timeline under one canonical contact, that a duplicate webhook is de-duplicated, and that adding a brand-new "matrix" channel needs **no schema change**.

9.2 How it reduces the invention to practice

The harness exercises every required behaviour: single-store write across channels (R1–R3), no-migration new channel (R4), canonical resolution (R5), contact-anchored timeline (R6–R7), idempotent ingest (R8), and metadata preservation/query (R9). Running `node src/demo.js` prints the unified timeline and exits non-zero if any invariant fails — an executable proof of enablement.

9.3 Configuration points

`unified-history.js` reads a config object: `channels` (the set), `normalize` (per-channel identifier normaliser), `timelineWindow` (default page size), `refreshDeliveryOnDuplicate` (bool). No identifiers, secrets, or infra names are embedded; everything behavioural is configuration.

---

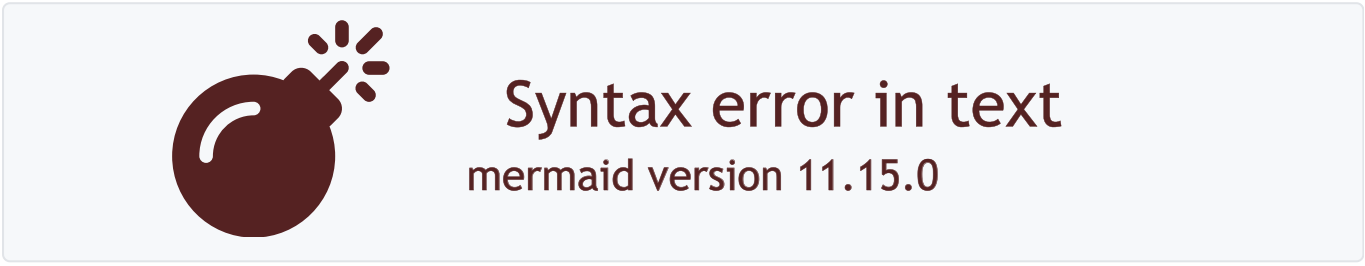
10. Worked Example / Scenario

Setup. Agent on WhatsApp + Telegram. Dana = `+15551234567` (WhatsApp) and `@dana` (Telegram), already linked to canonical `contact_id = 42`.

1. **08:10** — WhatsApp inbound from `+15551234567`: "Where's invoice INV-991?" `append` normalises the phone, `ĸ` resolves `contact_id=42`, stores row with `metadata={template:null}`.
2. **08:10** — Agent reads `timeline(42)`, replies on WhatsApp; outbound row appended with WhatsApp delivery metadata.

3. **14:02** — Telegram inbound from `@dana`: "any update on what I asked earlier?" ʞ resolves the **same** `contact_id=42`.
4. **14:02** — Agent reads `timeline(42)` — which now interleaves the **morning WhatsApp invoice exchange** with the Telegram message in time order — and answers with full context ("INV-991 is paid; receipt sent"). The channel switch was invisible.
5. **14:02** — A duplicate Telegram webhook for the same `provider_message_id` arrives; `append` is a no-op (idempotent).

10.1 Sequence diagram



---

11. Security, Safety & Failure Modes

11.1 Failure-mode table

Mode	Cause	Posture	Mitigation
Duplicate webhook	At-least-once delivery / retry	Fail-safe	Idempotent upsert on `(channel, provider_message_id)` (R8)
Malformed metadata	Provider sends non-JSON / unexpected shape	Fail-open (never drop)	Wrap raw payload `{_raw:…}`; message still stored (R12)
Resolver miss	First contact on a channel	Fail-open	Mint new canonical contact; link for future (R5)
Wrong merge (over-link)	Two humans share an identifier form	Fail-closed on merges	Merges are explicit/auditable; probabilistic links only *proposed* (§7.3)
Provider outage	One channel down	Isolated	Other channels write; timeline reads never fail on a down provider (R12)
Hot-partition reads	Very long contact timelines	Bounded	Keyset/windowed pagination on `(contact_id, occurred_at)` (R6)
Metadata index bloat	Unbounded JSON keys	Bounded	Index only allowlisted hot keys; GIN with `jsonb_path_ops`

11.2 STRIDE-style sketch

Threat	Concern	Control
Spoofing	Forged provider-native identifier	Trust boundary at adapter; canonical id never derived from unverified claims for *merges*
Tampering	Edit history to hide context	Append-only store; corrections are new rows (R10)
Repudiation	"I never sent that"	Append-only + provider_message_id + occurred/received times
Info disclosure	Cross-contact leakage in timeline	Timeline strictly filtered by one `contact_id`; per-tenant/user isolation at the read API
DoS	Webhook flood	Idempotency + rate limits at ingest (adapter layer)
Elevation	Reading another tenant's timeline	Authz on the read API; `contact_id` scoped to tenant

11.3 Privacy

Bodies and metadata may contain personal data; the store should support retention/erasure (delete-by-contact, channel-scoped erase) and minimisation (store only needed metadata keys). Erasure on an append-only store is handled by a tombstone + content redaction that preserves the audit shell. The design is amenable to GDPR/CCPA erasure (semantic alignment, §12) without breaking the timeline join.

---

12. Standards & Framework Mapping

We claim **semantic alignment**, not certified compliance.

Standard / framework	Relevance	Alignment
ISO 8601 / RFC 3339	Timestamps (`occurred_at`, `received_at`)	Store times as timezone-aware ISO 8601
E.164 (ITU-T)	Phone-number normalisation in κ	Phones normalised to E.164 for canonical resolution
RFC 5321/5322	Email identifiers in κ	Emails normalised (lower-case, trim) per addr spec
JSON / RFC 8259	`metadata` column encoding	Polymorphic metadata is RFC-8259 JSON in JSONB
GDPR / CCPA	Personal data in messages	Supports erasure-by-contact, minimisation, retention windows
OWASP ASVS	Read/write API hardening	Authz-scoped timeline reads; input validation at ingest
Event-sourcing / log-as-SoR	Append-only message log	Idempotent, append-only, replayable ingest
OpenTelemetry	Observability of ingest/read	Ingest and timeline spans are amenable to OTel tracing

No certification is asserted; mappings describe how the design *conforms in spirit* to widely-used standards.

---

13. Evaluation Methodology

Numbers below are **illustrative** placeholders for a real benchmark, not measured results; the value is the methodology.

Dimension	Metric	Method	Interpretation
Continuity	% of cross-channel handoffs where the agent had prior-channel context	Replay traced multi-channel conversations; check timeline coverage	Higher = less amnesia
Migration cost	DDL migrations needed to onboard N providers	Count schema changes adding providers 5→10	Target: **0** (the point)
Field fidelity	% of provider-specific fields preserved & queryable	Diff stored `metadata` vs source payloads	Higher = less field-loss tax
Identity recall	% of same-human cross-channel pairs resolved to one contact	Labelled identity set vs resolver output	Higher = fewer split contacts
Read latency	p50/p95 of `timeline(k)`	Load test with windowed reads on indexed `(contact_id, occurred_at)`	Lower = better agent loop
Idempotency	Duplicate-suppression rate under replayed webhooks	Inject N% duplicates; count stored dupes	Target: 0 stored dupes

Illustrative target: 0 migrations to add providers 5→10; ≥99% metadata fidelity; ≥98% identity recall on deterministically-linkable pairs; p95 `timeline` < 30 ms on indexed reads for typical contact histories. **All numbers illustrative.**

---

14. Novelty & Inventive Claims

Disclosed as prose; each narrows on a distinct novel feature. (Published defensively — these establish prior art, not exclusivity.)

Claim 1 (independent). A method for unified storage of agent–contact messaging interactions across a plurality of messaging providers, comprising: storing each interaction in a single relational table having at least a channel-identifier column, a contact-identifier column, a direction column, and a metadata column of polymorphic structured type; resolving, from a provider-native contact identifier, a canonical contact identity; and surfacing a per-contact timeline by selecting, across said channel-identifier column, the interactions whose resolved canonical contact identity matches a given contact, ordered by interaction time, as a single conversation thread supplied to an AI agent as conversation memory.

Claim 2. The method of claim 1, wherein provider-specific fields that are not common across said plurality of providers are stored in said metadata column such that onboarding a further provider with a different set of provider-specific fields requires **no** alteration of the table schema.

Claim 3. The method of claim 1, wherein resolving the canonical contact identity comprises normalising the provider-native identifier by a per-channel rule and looking the normalised identifier up in an identity-link table that maps (channel, normalised-identifier) to a canonical contact key.

Claim 4. The method of claim 3, wherein on a miss in the identity-link table a new canonical contact key is minted and a link inserted, so that subsequent interactions on that channel resolve to the same canonical

contact.

Claim 5. The method of claim 1, wherein storing each interaction is idempotent with respect to a key comprising the channel identifier and a provider message identifier, such that duplicate deliveries of the same interaction are stored at most once.

Claim 6. The method of claim 5, wherein, absent a provider message identifier, a deterministic surrogate identifier is derived from the channel, contact, time, and a hash of the interaction body for use in said idempotency key.

Claim 7. The method of claim 1, wherein the single table is append-only and corrections to a stored interaction are represented as new rows referencing the original via the metadata column rather than by destructive update.

Claim 8. The method of claim 1, wherein the metadata column is a JSON-document column that is queryable by JSON path and indexable on selected keys without promoting those keys to table columns.

Claim 9. The method of claim 1, wherein the per-contact timeline interleaves interactions from two or more distinct messaging providers in time order such that a change of provider by the contact is not reflected as a discontinuity in the timeline.

Claim 10. The method of claim 1, wherein the AI agent, prior to replying on any provider, reads the per-contact timeline assembled across all providers and supplies a recent window of it to a language model as conversation memory.

Claim 11. The method of claim 1, wherein the set of providers and the per-channel identifier-normalisation rules are supplied as external configuration rather than embedded in the storage schema or query.

Claim 12. The method of claim 1, wherein the per-contact timeline read is windowed by keyset pagination over a composite ordering of canonical contact key and interaction time.

Claim 13. The method of claim 1, wherein two canonical contact identities determined to denote the same person are merged by re-pointing their identity links to a single canonical contact key, the merge being recorded as an auditable operation.

Claim 14. The method of claim 1, wherein malformed provider metadata that is not valid JSON is preserved by wrapping it in a JSON document under a reserved key, so that no interaction is dropped on account of metadata shape.

Claim 15. The method of claim 1, wherein an outage of one provider does not prevent storage of interactions from other providers nor reads of the per-contact timeline, the timeline simply omitting interactions from the unavailable provider.

Claim 16. A system comprising a processor and storage configured to perform the method of any of claims 1–15, including the single polymorphic-metadata message table, the identity-link table, and a read interface returning the contact-anchored cross-channel timeline.

---

15. Limitations & Threats to Validity

- **Narrow novelty.** Unified inboxes and JSONB schemas are well-trodden; the inventive margin is the *combination* (migration-free metadata + canonical-contact timeline + agent-memory framing). We state this plainly (US ~48/100, DE ~35/100 in internal assessment) — hence defensive publication, not filing.
- **Identity resolution is the hard part.** Deterministic linking misses humans who never share a verifiable identifier across channels; probabilistic stitching risks over-merging. We bound this by keeping merges explicit and auditable, but recall is inherently incomplete.
- **JSONB is not free.** Heavy unindexed JSON filters can be slow; metadata index bloat is possible. Mitigated by allowlisted hot-key indexing.
- **Append-only vs erasure tension.** Privacy erasure on an append-only store requires tombstoning/redaction; we describe but do not fully specify a retention engine here.
- **Withheld.** Provider-adapter transport details, production hardening, tenant isolation specifics, and proprietary deployment identifiers are intentionally out of scope and not disclosed.
- **Where prior art is closest.** CRM unified inboxes (read model) and CDP identity resolution (the join) are the nearest; our delta is in §5.

---

16. Future Work & Open-Source Reference App

Planned: a small open-source **unified multi-channel message store + contact timeline** service (Node + PostgreSQL) exposing `append` and `timeline`, with pluggable provider adapters and a configurable resolver. Roadmap: (1) the store + resolver + timeline; (2) idempotent ingest + duplicate replay tests; (3) JSONB metadata indexing helpers; (4) erasure/retention engine; (5) an AKS deployment chart. See **docs/OPEN-SOURCE-APP.md** for the concept, architecture, invention mapping, and a generic Kubernetes/AKS deployment sketch.

---

17. Conclusion

Cross-channel conversation continuity for an AI agent does not require a federation of stores or a migration per provider. A **single table of cross-channel invariants plus one polymorphic metadata column**, read through a **contact-anchored timeline view** keyed on a **canonical contact identity**, gives the agent one continuous history per contact — across WhatsApp, Telegram, Signal, Discord, and whatever comes next — with **zero schema migration** to add a channel and **no field loss** for the channels already onboarded. We publish this design defensively so it stays free to practice. The accompanying clean-room reference reduces it to practice; the prior-art delta states honestly what is new.

Appendix A — Prior-Art Landscape (well-trodden vs candidate-novel)

Well-trodden (NOT claimed as novel): unified-inbox CRM timelines; JSONB / document-in-relational / EAV / single-table storage; CDP deterministic identity stitching; messaging-gateway normalized envelopes; append-only event logs; chat-memory libraries.

Candidate-novel (the disclosed combination): the *migration-free polymorphic provider-metadata column* used specifically for messaging; the *canonical-contact timeline join* that makes "the conversation" a property of the contact not the channel; and the *agent-memory framing* in which that one timeline is the model's working memory. The novelty is the combination, not any single ingredient.

Honesty attestation. The prior-art search underlying this document is **directional, not exhaustive**. It reflects products and literature known to the author as of 2026-06-25. No claim is made that every relevant reference has been found. This publication exists to *establish* prior art, and readers are invited to cite additional art. We do not assert that the combination is patentable — internal assessment rates it marginal — which is precisely why it is published defensively.

Appendix B — Glossary

- **Channel / provider** — a messaging service (WhatsApp, Telegram, Signal, Discord, …).
- **Cross-channel invariant** — a field present for every provider (channel, contact, direction, body, time), promoted to a typed column.
- **Polymorphic metadata** — provider-specific fields stored in one JSONB column, schemaless at the column level.
- **Canonical contact** — the single identity to which a human's per-channel identifiers resolve.
- **Identity link** — a `(channel, identifier) → contact_id` mapping.
- **Timeline** — the time-ordered union of one canonical contact's interactions across all channels.
- **Idempotent append** — insert keyed by `(channel, provider_message_id)` so duplicates are no-ops.
- **Migration-free** — adding a provider needs no DDL change.

Appendix C — Reference-Implementation Index

File	Role
`src/schema.sql`	DDL for `contacts`, `identity_links`, `messages` + idempotency/timeline indexes
`src/unified-history.js`	In-memory store: `append`, resolver `resolve`, `timeline`; externalised config
`src/demo.js`	Self-checking worked example (Dana cross-channel) with assertions
`src/README.md`	What it shows, how to run, clean-room/illustrative disclaimer

Appendix D — Defensive-Publication Deposit & Timestamp

- **Publication date:** 2026-06-25
- **Publisher / copyright holder:** Gus IT LLC (Florida, USA)
- **Author:** Gustavo Assuncao, PhD
- **Version:** 1.0
- **Deposit channel:** to be assigned (IP.com / Zenodo / arXiv) — establishes a public, dated, citable prior-art record. No DOI is asserted by this draft.

> **This disclosure is intentionally public to bar later patenting of the disclosed subject matter by others.** It is published so that the techniques herein remain free for anyone to practice.

> The AGPL-3.0-or-later license adds an express patent grant for the accompanying reference implementation.