**AssistedDiscovery — Comprehensive Design Document**

Prepared for review and presentation · 26 Sep 2025, 05:36

# 1. Executive Summary

AssistedDiscovery provides deterministic, token‑efficient extraction and pattern learning over airline NDC XMLs to power discovery of reusable business patterns and identification/classification in new files. The system streams XML with lxml.iterparse, targets only high‑signal sections per (spec\_version, message\_root), emits compact PII‑masked NodeFacts, and uses micro‑batched LLM calls for pattern discovery and classification. This document details architecture, data model, flows (Discovery & Identify), performance, security, implementation plan, and testing.

# 2. Problem Statement & Goals

Problems Today  
- Non‑deterministic pattern counts across runs; unstable outputs.  
- Token‑limit failures on large XMLs due to whole‑document prompting.  
- Unnecessary pattern generation for low‑signal nodes.  
- Identify path also hits token limits on huge inputs.

Goals  
- Deterministic, repeatable extraction using curated targets per spec version & message root.  
- Token‑efficient streaming parse; micro‑batched LLM calls (3–6 facts).  
- Stable pattern catalog via de‑duplication (signature\_hash).  
- Identify pipeline that retrieves Top‑K candidates cheaply, then classifies via LLM with hard validations.

Non‑Goals (MVP)  
- Full XSD validation of every NDC version.  
- Embeddings‑based retrieval (may be added later).  
- Cross‑message orchestration beyond a single XML run.

# 3. High‑Level Architecture

Core components: API, Job Orchestrator, XML Stream Parser, Extractors (template or generic LLM), LLM Gateway, Ranker/Retriever, Dedup/Versioning, Report Builder, and MySQL storage (Catalog + Facts/Patterns).

Conceptual Diagram:  
[Client/UI] -> [API] -> [Jobs] -> [Parser(iterparse)] -> [Extractors]  
 | |  
 [Catalog/Targets] [NodeFacts -> MySQL]  
 [LLM Gateway] <-> Discovery (Patterns) [Dedup -> Patterns]  
 [LLM Gateway] <-> Identify (Classify) [Matches -> Report]

# 4. End‑to‑End Flow

Startup (per process)  
1) Load catalog: ndc\_target\_paths and ndc\_path\_aliases into memory; compile constraints/regex.  
2) Build a path‑trie of local‑name segments for fast end‑event matching.

Common Steps (per file upload)  
1) Detect message\_root and spec\_version from root tag/namespace.  
2) Resolve targets; apply alias fallback if exact version missing.  
3) Stream XML with iterparse; when stack path matches a target, yield subtree and clear for O(1) memory.  
4) Extract NodeFacts: PII‑masked summaries with ids/refs; persist to MySQL; build in‑run ID indexes and optional AssociationFacts.

Discovery  
1) Group NodeFacts per section and micro‑batch (3–6) into LLM with strict JSON schema.  
2) Normalize candidate Patterns and compute signature\_hash for de‑dup; upsert to patterns; track times\_seen and examples.

Identify  
1) For each NodeFact, retrieve Top‑K candidate patterns by cheap similarity (path match + children/attrs Jaccard, etc.).  
2) Apply hard constraints (decision\_rule) server‑side to filter candidates.  
3) LLM classify among survivors (or none) and store pattern\_matches with confidence.  
4) Compile a Gap Report: coverage by importance, missing required sections, constraint violations, unmatched nodes, dangling refs.

# 5. Data Model (MySQL)

Tables and key columns:  
ndc\_target\_paths(  
 id, spec\_version, message\_root, path\_local, extractor\_key,  
 is\_required, importance, constraints\_json, notes  
) -- unique(spec\_version, message\_root, path\_local)

ndc\_path\_aliases(  
 id, from\_spec\_version, from\_message\_root, from\_path\_local,  
 to\_spec\_version, to\_message\_root, to\_path\_local,  
 is\_bidirectional, reason  
)

runs(id, kind, started\_at, finished\_at)

node\_facts(  
 id, run\_id, spec\_version, message\_root, section\_path,  
 node\_type, node\_ordinal, fact\_json(JSON), pii\_masked, created\_at  
)

association\_facts(  
 id, run\_id, rel\_type, from\_node\_fact\_id, to\_node\_fact\_id,  
 from\_node\_type, to\_node\_type, ref\_key, created\_at  
)

patterns(  
 id, spec\_version, message\_root, section\_path,  
 selector\_xpath, decision\_rule(JSON), signature\_hash, times\_seen,  
 created\_by\_model, examples(JSON), created\_at  
) -- unique(signature\_hash)

pattern\_matches(  
 id, run\_id, node\_fact\_id, pattern\_id, confidence, verdict, created\_at  
)

# 6. Version & Namespace Handling

- Prefer @Version on root when present (e.g., Version='17.2').  
- Fallback: derive from namespace URI segment (…/2018.2/… → 18.2).  
- Normalize vendor prefixes in message\_root (e.g., 'IATA\_OrderViewRS' → 'OrderViewRS').  
- Alias fallback allows minor version reuse (e.g., 18.2 → 17.2) when targets are compatible.

# 7. Parsing & Matching

iterparse Strategy  
- Single pass, 'start'/'end' events; maintain local‑name stack.  
- On 'end', compare absolute local‑name path to path‑trie/lookup; if match, yield and clear subtree to bound memory.

High‑Signal Path Trie  
- Store each target path as a tuple of segments; keep a set of leaf signatures.  
- Allows O(1) average evaluation per 'end' event with minimal overhead.

# 8. Extraction Approaches

Template‑Driven Extractors  
- Small JSON spec per section describes repeat nodes, fields, attrs, masks, ids/refs; generic engine emits NodeFacts.  
- Pros: fast, deterministic, cheapest tokens. Cons: needs curation per section.

Generic LLM Extractor  
- Send subtree XML + fixed NodeFact schema; model emits NodeFacts with masking and structure.  
- Pros: zero per‑node code, handles unknown XML. Cons: token cost; must cap subtree length (e.g., 3–4 KB); strict JSON validation.

Both can co‑exist: prefer template where available; fallback to generic LLM where not.

# 9. Pattern Learning (Discovery)

- Batch 3–6 NodeFacts per call; temperature=0, top\_p=0; strict JSON schema to return Patterns (selector\_xpath + decision\_rule).  
- Normalize: sort child lists, canonicalize XPath, strip whitespace.  
- signature\_hash = sha256(json\_dumps({root, version, section\_path, selector, rule}, sort\_keys=True)).  
- Upsert by signature\_hash; increment times\_seen; store masked examples for explainability.

Pattern (JSON):  
{  
 "section\_path": "/OrderViewRS/Response/Order/BookingReferences",  
 "selector\_xpath": "./BookingReference[ID]",  
 "decision\_rule": {  
 "must\_have\_children": ["ID"],  
 "optional\_children": ["AirlineID","OtherID"],  
 "attrs": {}  
 }  
}

# 10. Identify (Retrieval + Classify)

Top‑K Retrieval (cheap)  
- Filter candidates by (message\_root, section\_path prefix).  
- Score = α·path\_match + β·Jaccard(children∪attrs) + γ·flags(ids\_present, refs\_present).  
- K=5 is sufficient for MVP; index patterns on (message\_root, section\_path).

Hard Validations  
- Apply decision\_rule before LLM: must\_have\_children, required attrs, regex constraints\_json.

LLM Classifier  
- Prompt with NodeFact + 3–5 candidate pattern summaries; return {pattern\_id|none, confidence ∈ [0,1]}.  
- Store into pattern\_matches; threshold (e.g., 0.7) to mark definitive matches.

Gap Report  
- Coverage by importance; list missing required sections; unmatched NodeFacts; constraint violations; dangling refs.

# 11. Security & PII

- Mask emails (\*\*\*@\*\*\*), phones (keep last 4 digits), dates (YYYY‑\*\*‑\*\*), long numeric IDs (keep last 2–4).  
- Run a server‑side PII gate on fact\_json before insert (regex).  
- Restrict snippets to ≤120 chars; never store full unmasked documents in NodeFacts.  
- Store raw XML only in object store with access controls; never send entire files to LLM.

# 12. Performance & Scaling

- iterparse is O(|XML|); memory is O(depth).  
- Targeted sections drastically reduce tokens; micro‑batches cap per‑call tokens.  
- Throughput estimate (MVP): ~5–15 MB XML/min per worker on commodity VM; LLM calls dominate latency.  
- Horizontal scale by workers; concurrency per file limited by sections discovered.  
- Caching: HS targets, alias maps, compiled constraints; refresh every N minutes or via webhook.

# 13. Observability & Operations

- Metrics: runs/s, tokens in/out, LLM latency, NodeFacts/sec, patterns emitted, match confidence histogram, coverage%.  
- Logs: per section\_path decisions, LLM request IDs, errors with section context.  
- Audit: runs table links all artifacts; store prompts/responses in redacted form for replay.  
- Feature flags: switch between template and generic LLM extractor per section.

# 14. Failure Modes & Handling

- Malformed XML → abort run with parse error and pointer to byte offset.  
- Token limit → shrink subtree window or switch to template extractor; reduce batch size.  
- Empty targets → alias fallback; if still empty, use generic ‘significant subtree’ heuristic.  
- LLM JSON invalid → retry once with repair prompt; else skip batch and log.

# 15. API Surface (MVP)

POST /v1/runs?kind=discovery|identify (multipart XML upload or object key)  
GET /v1/runs/{run\_id} (status, metrics)  
GET /v1/runs/{run\_id}/report (discovery/identify report)  
GET /v1/node\_facts?run\_id=...&section=...  
GET /v1/patterns?message\_root=...&section=...

# 16. Implementation Plan (1 week MVP)

Day 1–2  
- DB schema, models, and seed ndc\_target\_paths + aliases for OrderViewRS 17.2.  
- Version/root detection; iterparse streamer; path‑trie matcher; basic extractors (template or generic).

Day 3  
- NodeFacts persistence with PII gate; AssociationFacts for basic refs; caching layer.

Day 4  
- Discovery micro‑batch prompt + JSON schema; de‑dup (signature\_hash); persistence of patterns.

Day 5  
- Identify pipeline: Top‑K retrieval, hard checks, classify prompt; pattern\_matches; Gap Report.

Day 6  
- Metrics/logging, retries, error surfaces; golden tests on provided sample XML.

Day 7  
- Docs, runbooks, and a small Analyst UI page for upload & report; dry‑run with 2–3 real samples.

# 17. Testing Strategy

- Unit: version detection, path‑trie matching, masking functions, signature canonicalization.  
- Integration: run Discovery/Identify against known XML fixtures; assert deterministic counts and hashes.  
- Golden: freeze LLM responses with fixtures (or VCR) for CI determinism.  
- Load: large XML with many sections; ensure memory remains bounded and throughput acceptable.

# 18. Risks & Mitigations

- Spec drift across airlines: maintain ndc\_path\_aliases and constraints; add per‑airline overrides if needed.  
- LLM variability: temp=0, strict JSON schema, retry with repair; de‑dup by signature.  
- PII leak: dual masking (prompt + server gate); limit snippet length; redact logs.  
- Token cost: prefer template extractor; cap subtree size; batch tightly.

# 19. Appendices

A. Example NodeFact  
{  
 "spec\_version":"17.2","message\_root":"OrderViewRS",  
 "section\_path":"/OrderViewRS/Response/DataLists/PassengerList",  
 "node\_type":"Passenger","node\_ordinal":1,  
 "children":["PTC","Birthdate","Individual","ContactInfoRef"],  
 "attrs":["PassengerID"],  
 "code\_values":{"PTC":"ADT"},  
 "ids":{"PassengerID":"T1"},  
 "refs":{},  
 "snippet":"<Passenger PassengerID=T1> PTC=ADT Birthdate=1990-\*\*-\*\*",  
 "pii\_masked":true,  
 "values":{"PTC":"ADT","Birthdate":"1990-\*\*-\*\*","Gender":"Male"}  
}

B. Discovery Prompt Skeleton (tool-call)  
System: You produce Patterns from NodeFacts. Return strict JSON.  
User:  
Context: spec\_version=17.2, message\_root=OrderViewRS, section\_path=/.../BookingReferences  
Here are 4 NodeFacts (masked). Emit <=2 Patterns with selector\_xpath + decision\_rule.  
-- facts JSON here --  
Tool: emit\_patterns({ "patterns":[ ... ] })

C. Identify Classifier Prompt Skeleton (tool-call)  
System: You classify NodeFact against candidate Patterns.  
User:  
NodeFact: {...}  
Candidates: [{pattern\_id, selector\_xpath, decision\_rule\_summary}, ...]  
Return: {"verdict":{"pattern\_id": "...", "confidence": 0.82}}

D. Top‑K Retrieval (cheap)  
SQL sketch:  
SELECT p.\*,  
 (CASE WHEN p.section\_path = :nf\_section THEN 1 ELSE 0 END) \* 0.5 +  
 jaccard(p.children\_sig, :nf\_children\_sig) \* 0.4 +  
 (CASE WHEN :nf\_has\_ids THEN 0.1 ELSE 0 END) AS score  
FROM patterns p  
WHERE p.message\_root = :root AND p.section\_path LIKE :prefix  
ORDER BY score DESC LIMIT 5;

E. Signature Canonicalization (pseudo)  
normalize\_xpath(x): strip spaces; sort predicates; collapse '//' where safe  
canonicalize\_rule(r): sort keys; sort arrays; drop example text  
signature = sha256(json.dumps({root,version,section,xpath,r}, sort\_keys=True))