

QueryGraph Architecture

Semantic Infrastructure for Agentic AI

AI Navigator semantic layer in Rust

Semantic Croissant · CDIF · DID · ODRL

Report scope

This document explains the meaning, architecture, and Rust implementation of QueryGraph as a four-layer semantic system for governed dataset discovery and agentic AI.

Implementation

Rust crate with CLI, JSON-LD model builders, DID generation, ODRL policy model, and CODATA DID anchoring client.

Generated from

docs/querygraph-architecture-report.md using Pandoc and Typst.

Contents

QueryGraph Architecture: Semantic Infrastructure for Agentic AI	3
Abstract	3
Why QueryGraph Exists	3
Reference Landscape	3
Rust Codebase Structure	5
The Four-Layer Bundle	6
CODATA ODRL Demo Reproduction	7
Meaning of the Architecture	7
Why Rust	7
Tests and Verification	8
Limitations	8
Roadmap	8
Conclusion	9
References	9

QueryGraph Architecture: Semantic Infrastructure for Agentic AI

Abstract

QueryGraph is a Rust codebase for building an AI Navigator semantic layer: a machine-readable contract between datasets, knowledge graphs, autonomous agents, decentralized identity, and digital rights policy. The first implementation composes four layers into a single JSON-LD bundle:

1. Semantic Croissant for dataset metadata.
2. CDIF for cross-domain FAIR interoperability.
3. DID for decentralized identity and provenance.
4. ODRL for machine-actionable rights and policy.

The codebase is intentionally small. It does not try to implement a full catalog, graph database, model runtime, or user interface yet. Instead, it implements the semantic core that those systems need to trust, discover, govern, and reuse data.

Why QueryGraph Exists

The practical problem is that AI agents need more than files and URLs. They need to know what a dataset means, how it is structured, whether it can be used, who is responsible for it, and what policies constrain downstream actions such as indexing, transformation, translation, or model training.

QueryGraph treats those questions as architecture, not metadata decoration. The semantic layer is the first-class interface between:

- datasets and dataset catalogs,
- knowledge graphs and vector indexes,
- agents and local tool runners,
- provenance systems and decentralized identifiers,
- rights policies and enforcement points.

This direction matches the public QueryGraph.ai resource spine, which points to DID Core, ODRL, the CODATA ODRL/DID demo, DDI ISO, Croissant, CODATA, CR4AI, and talks about Croissant, CDIF, metadata enhancement, and living knowledge graphs.

Reference Landscape

QueryGraph Resources

The QueryGraph resources page is the architectural reading list for this implementation. It links DID Core, ODRL Information Model, the CODATA ODRL and DID demo, ODRL support in Ollama, DDI ISO Standard, Croissant 1.0, Croissant 1.1, CODATA, CR4AI, and talks on Croissant/CDIF and metadata for AI readiness. In the Rust implementation, that page is not only background reading. The CLI can reproduce the CODATA demo flow for the resources page:

```
cargo run -- anchor-url --url "https://querygraph.ai/resources/"
```

That command calls the live endpoint:

```
https://odrl.dev.codata.org/api/did/create_from_url
```

During verification, the endpoint returned a `did:oyd:zQm...` identifier and a stored payload with the page title `Resources`.

DID Core

DID Core defines decentralized identifiers as identifiers associated with DID documents, controllers, verification material, and service endpoints. QueryGraph uses this model for attribution: every semantic bundle can identify the service or agent responsible for producing it.

The local implementation in [src/did.rs](#) creates deterministic `did:oyd`-style identifiers from a seed. It uses SHA-256, wraps the digest in a multihash prefix, and encodes it with base58btc so generated identifiers match the `zQm...` form returned by the CODATA ODRL demo.

This is not a replacement for a production DID wallet. It is a stable local identity surface that can later be replaced by a wallet-backed resolver without changing the surrounding semantic bundle structure.

ODRL Information Model

ODRL defines a policy expression model for permitted and prohibited actions over assets, with support for constraints, duties, obligations, and policy profiles. QueryGraph uses ODRL as the policy layer attached to a dataset target.

The current implementation in [src/odrl.rs](#) models:

- Policy
- Rule
- Action
- permissions
- prohibitions
- simple action checks with `Policy::allows`

The first policy generated by the navigator permits public reading with attribution, permits local indexing for the agent DID, and prohibits public derivative use for model training unless a separate agreement exists.

The next obvious expansion is richer ODRL support: duties, consequences, remedies, logical constraints, profile inheritance, and conflict strategies.

Croissant

MLCommons Croissant is the dataset metadata layer. Croissant 1.1 is especially relevant because it focuses on agent-ready metadata, provenance, vocabulary interoperability, and governance.

QueryGraph's Croissant implementation is in [src/croissant.rs](#). It defines:

- `CroissantDataset`
- `FileObject`
- `RecordSet`
- `Field`

The output is JSON-LD using Schema.org and Croissant vocabulary terms. The navigator creates a basic dataset with one source file and a default record set with semantic fields:

- `subject` mapped to `https://schema.org/about`
- `value` mapped to `https://schema.org/value`
- `source` mapped to `https://schema.org/citation`

This is enough to demonstrate the key idea: an AI system should inspect a dataset's machine-readable structure before loading, indexing, translating, or deriving from it.

CDIF

CDIF, the Cross-Domain Interoperability Framework, is the interoperability layer for FAIR data across domains. QueryGraph uses CDIF to project Croissant metadata into cross-domain discovery and reuse concerns rather than trapping metadata in an ML-only shape.

The implementation is in [src/cdif.rs](#). It models five profile concerns:

- Discovery
- Data Access
- Controlled Vocabularies
- Data Integration
- Universals

`CdifResource::from_croissant` takes a Croissant dataset and creates a DCAT-style CDIF JSON-LD projection with landing page, access service, vocabulary references, units, and optional temporal and spatial coverage.

DDI ISO

The DDI Alliance page reports that DDI was recognized as ISO/PAS 25955:2026, based on DDI Common Core. The relevance for QueryGraph is the research and statistical metadata lineage: variables, units, value domains, and data lifecycle concepts should become first-class semantic objects over time.

This first Rust implementation does not yet model DDI directly. It leaves room for a `ddi` or `variables` module that can represent DDI-style variables and connect them to CDIF universals and Croissant fields.

Pale Fire

AgStack Pale Fire is relevant because it frames knowledge graph search as a combination of semantic relevance, connectivity, temporal matching, query matching, and entity-type reasoning. QueryGraph does not yet implement graph search, but its semantic bundle is designed to be the upstream substrate for such ranking. The implementation therefore keeps the metadata typed and graph-friendly. Each layer emits JSON-LD identifiers and vocabulary references that can be loaded into a graph store or converted into RDF later.

Croissant Toolkit, Agents CLI, The Minority Report, and ODRL Ollama

The CODATA Croissant Toolkit demonstrates a practical skill ecosystem around Croissant, ODRL, CDIF, RO-Crate, Neo4j, and semantic dataset search. The agents-cli local development guide shows how agents can be run locally with skill loading, evaluation, linting, and playground workflows. The Minority Report demonstrates multilingual controlled vocabulary generation and benchmarking, with formal provenance in Croissant metadata. CODATA's ODRL support in Ollama demonstrates server identity, DID attribution, and policy-aware local AI services.

QueryGraph borrows the architecture, not the implementation:

- Use Croissant for formal dataset metadata.
- Use CDIF for cross-domain interoperability.
- Use DIDs for agent and service identity.
- Use ODRL for action governance.
- Keep local CLI flows reproducible.
- Make output suitable for graph indexing and agent inspection.

Rust Codebase Structure

The codebase currently has one binary crate and one library surface:

```
src/
  lib.rs
  main.rs
  croissant.rs
  cdif.rs
  did.rs
  odrl.rs
  codata.rs
  navigator.rs
```

Library Entry Point

`src/lib.rs` exposes the modules and re-exports the main navigator types:

```
pub mod codata;
pub mod cdif;
pub mod croissant;
pub mod did;
pub mod navigator;
pub mod odrl;

pub use navigator::{AiNavigator, NavigatorInput, NavigatorOutput};
```

The library boundary matters because QueryGraph should be usable from a CLI, a server, a graph ingestion job, or an agent runtime.

CLI

`src/main.rs` uses `clap` and provides two commands:

```
querygraph navigator
querygraph anchor-url
```

`navigator` builds the four-layer semantic bundle:

```
cargo run -- navigator \
  --dataset-name "Hazard vocabulary" \
  --description "Controlled vocabulary with multilingual technical terms" \
  --landing-page "https://querygraph.ai/datasets/hazards" \
  --data-url "https://querygraph.ai/datasets/hazards.csv"
```

`anchor-url` calls the CODATA ODRL demo API:

```
cargo run -- anchor-url --url "https://querygraph.ai/resources/"
```

AI Navigator Composition

`src/navigator.rs` is the composition root. `AiNavigator::build` takes `NavigatorInput` and produces `NavigatorOutput`.

The output contains:

- `generated_at`
- `croissant`
- `cdif`
- `did`
- `odrl`
- `bundle`

The bundle is the article-worthy core of the architecture. It is a JSON-LD object of type:

```
querygraph:AiNavigatorSemanticBundle
```

It contains a compact `@context` with Schema.org, Croissant, CDIF, DCAT, DCTerms, ODRL, and QueryGraph namespaces.

The Four-Layer Bundle

Layer 1: Semantic Croissant

The navigator first creates a `CroissantDataset`. This is the view a dataset catalog, ML tool, or agent should read before touching data.

Key concepts:

- identity: `@id`
- human meaning: `name`, `description`, `keywords`
- legal metadata: `license`
- provenance: `creator`
- bytes: `distribution`
- structure: `recordSet`
- semantics: field-level `sameAs`

The generated Croissant JSON-LD currently uses a small default schema. That is deliberate. The immediate goal is not complete domain modeling; it is a reliable scaffold that can be enriched by parsers, catalog importers, or DDI/CDIF variable services.

Layer 2: CDIF

The CDIF layer is generated from the Croissant dataset. This makes CDIF a projection over dataset metadata instead of a separate hand-authored document.

The generated CDIF layer includes:

- dataset ID
- CDIF profiles
- landing page
- access service
- temporal coverage
- spatial coverage
- units
- controlled vocabularies

This is the layer that should make QueryGraph useful outside a single domain.

Layer 3: DID

The DID layer identifies the agent or service that generated the bundle.

The current local DID algorithm:

1. Build a seed from agent name, creator, and dataset name.
2. Hash with SHA-256.
3. Prefix the digest as a SHA-256 multihash.
4. Encode with base58btc.
5. Emit `did:oyd:zQm...`

The DID document also includes DID Core and Ed25519 context URLs and can carry a service endpoint.

This gives QueryGraph deterministic local attribution today and a clean upgrade path to real wallet-backed DIDs later.

Layer 4: ODRL

The ODRL layer attaches policy to the dataset target.

The default generated policy says:

- public may read with attribution,
- the agent DID may index locally,
- public may not derive for model training without a separate agreement.

That default is intentionally opinionated: AI systems should not treat dataset availability as permission for every downstream use.

CODATA ODRL Demo Reproduction

The CODATA ODRL demo at <https://odrl.dev.codata.org/dids?ref=querygraph.ai> is a Vite/React application. Its frontend uses an Axios client with base URL `/api`. Inspecting the bundled JavaScript showed these API paths:

- `/did/create`
- `/did/create_from_url`
- `/oac/policy`
- `/oac/search`
- `/variables/create`
- `/croissants/create`
- `/groups/create`

QueryGraph implements the relevant first reproduction path in [src/codata.rs](#):

```
pub fn create_did_from_url(&self, url: &str) -> Result<AnchoredDid, request::Error>
```

The verified command:

```
cargo run -- anchor-url --url https://querygraph.ai/resources/
```

returned a live response shaped as:

```
{
  "did": "did:oyd:zQmc1qjwR9FFbNU99U44oVVPt8FMahh8WJqPkGWagqvZe5Z",
  "doc": null,
  "stored_payload": {
    "url": "https://querygraph.ai/resources/",
    "timestamp": "2026-05-23T04:36:33.082502",
    "title": "Resources",
    "is_rdf": false
  }
}
```

That gives QueryGraph a concrete bridge to the CODATA DID infrastructure while keeping its own local identity generation available for offline use.

Meaning of the Architecture

QueryGraph's central claim is that AI navigation should be governed semantic navigation.

An agent should not simply retrieve a URL, embed a document, or call a model. It should ask:

- What is this dataset?
- What are its fields and record sets?
- Which vocabularies define its terms?
- Which cross-domain profiles does it satisfy?
- Who or what produced this metadata?
- What actions are permitted?
- What actions are prohibited?
- Can this be indexed?
- Can this be transformed?
- Can this be used for model training?

The Rust implementation answers those questions through typed structures that serialize to JSON-LD.

This is important because JSON alone is not enough. JSON describes shape. JSON-LD describes shape plus linked meaning. QueryGraph's bundle is meant to survive movement between a local CLI, a web API, a knowledge graph, a dataset catalog, and an agent runtime.

Why Rust

Rust is a good fit for this layer because the semantic bundle should be reliable infrastructure:

- Types make policy and metadata structures explicit.
- Serialization is deterministic and testable.
- CLI tools can be distributed as static binaries.
- A future server can share the same library with the CLI.
- Rust can sit comfortably near graph engines, data pipelines, and local agent runtimes.

The current implementation uses:

- `serde` and `serde_json` for JSON and JSON-LD serialization.
- `clap` for CLI parsing.
- `chrono` for timestamps.
- `sha2` and `bs58` for deterministic DID generation.
- `request` for CODATA ODRL API calls.
- `anyhow` for CLI error handling.

Tests and Verification

The repository currently includes one unit test in [src/navigator.rs](#). It verifies that `AiNavigator::build` produces all four semantic layers:

- Croissant dataset
- CDIF dataset
- DID beginning with `did:oyd:zQm`
- ODRL policy

The implementation was also manually verified with:

```
cargo fmt
cargo test
cargo run -- navigator ...
cargo run -- anchor-url --url https://querygraph.ai/resources/
```

Limitations

This is a foundation, not a finished platform.

Current limitations:

- ODRL constraints are strings, not a full typed constraint expression model.
- Duties, consequences, remedies, obligations, and conflict strategies are not implemented yet.
- DID generation is deterministic and local, not wallet-backed.
- CDIF profile IRIs are provisional implementation IRIs, not a complete official CDIF vocabulary binding.
- Croissant support is a useful subset, not the full Croissant 1.1 standard.
- DDI and variable-domain modeling are not yet implemented.
- There is no graph database adapter yet.
- There is no RDF/Turtle export yet.
- There is no HTTP server or MCP server yet.
- There is no policy decision point beyond simple `allows` checks.

These limitations are acceptable for the first codebase because the most important early outcome is a coherent architecture that compiles and produces usable semantic bundles.

Roadmap

1. Full ODRL Policy Engine

Add typed support for:

- duties,
- obligations,
- consequences,
- remedies,
- logical constraints,
- temporal constraints,
- spatial constraints,
- assignee/assigner collections,
- policy inheritance,
- conflict strategy.

2. Croissant 1.1 Expansion

Add richer Croissant support:

- data resources,
- extraction rules,
- provenance,
- vocabulary mappings,
- governance metadata,

- validation against JSON Schema or SHACL.

3. CDIF and DDI Variable Layer

Add variables as first-class objects:

- names,
- labels,
- units,
- value domains,
- concepts,
- DDI Common Core alignment,
- CDIF universals.

4. Graph Export

Add exporters for:

- RDF/Turtle,
- N-Quads,
- Neo4j Cypher,
- Qdrant payloads,
- JSON-LD framed documents.

5. Agent Runtime Integration

Expose the semantic layer through:

- MCP server tools,
- agents-cli compatible local commands,
- policy-aware dataset inspection,
- policy-aware indexing actions,
- benchmark/evaluation harnesses.

6. Wallet-Backed DID Support

Integrate with:

- CODATA ODRL API,
- local `~/.odrl/did.json`,
- Universal Resolver,
- verifiable credentials,
- service endpoint discovery.

Conclusion

QueryGraph is best understood as semantic infrastructure for responsible agentic AI. It connects dataset meaning, cross-domain interoperability, decentralized identity, and machine-actionable rights into one typed Rust composition.

The implementation is small but significant: it makes the architecture executable. A user can run one command and get a four-layer JSON-LD bundle. A user can run another command and reproduce the CODATA URL-to-DID anchoring flow for QueryGraph resources. That is the beginning of a practical AI Navigator: not just a system that finds data, but one that knows what data means, where it came from, and what it is allowed to do.

References

- QueryGraph Resources: <https://querygraph.ai/resources/>
- W3C DID Core: <https://www.w3.org/TR/did-1.0/>
- W3C ODRL Information Model 2.2: <https://www.w3.org/TR/odrl-model/>
- CODATA ODRL/DID Demo: <https://odrl.dev.codata.org/dids?ref=querygraph.ai>
- CODATA ODRL API URL anchoring endpoint: https://odrl.dev.codata.org/api/did/create_from_url
- CODATA ODRL support in Ollama: <https://github.com/codata/ollama/blob/main/ODRL.md>
- DDI ISO Standard: <https://ddialliance.org/iso-standard>
- MLCommons Croissant 1.1: <https://mlcommons.org/2026/02/croissant-1-1-standard/>
- CODATA CDIF: <https://codata.org/initiatives/making-data-work/cdif/>
- AgStack Pale Fire: <https://github.com/agstack/palefire>
- CODATA Croissant Toolkit: <https://github.com/codata/croissant-toolkit>
- CODATA agents-cli local development: <https://github.com/codata/agents-cli/blob/main/local.md>
- CODATA The Minority Report: <https://github.com/codata/the-minority-report>
- The Minority Report benchmark notes: <https://github.com/codata/the-minority-report/blob/benchmarks/benchmark.md>