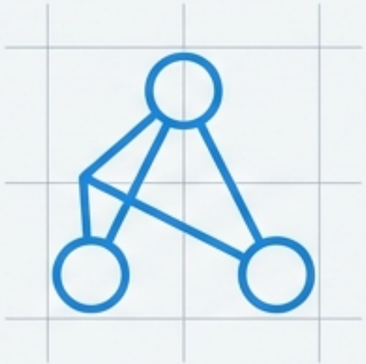
OPAL Agent

The OPAL Agent: A SPARQL-Driven Knowledge Graph Potocol

An Architectural Overview of a High-Fidelity Query Agent

spec sheet	
ID:	`my-new-basic-sparql-agent`
Name:	`Basic Generic SPARQL-Driven OPAL Agent`
Version:	`1.1.0`
Created:	`2025-10-17`

The Agent at a Glance: Core Capabilities



Features

- **Multi-Query Execution**: Executes SQL, SPARQL, SPASQL, and GraphQL queries against any target.
- **Ontology-Aware SPARQL**: Leverages shared ontologies to construct more accurate and contextually relevant SPARQL queries.
- **Data Source Exploration**: Intelligently explores data sources to identify entity types and their prevalence before querying.

Primary Function

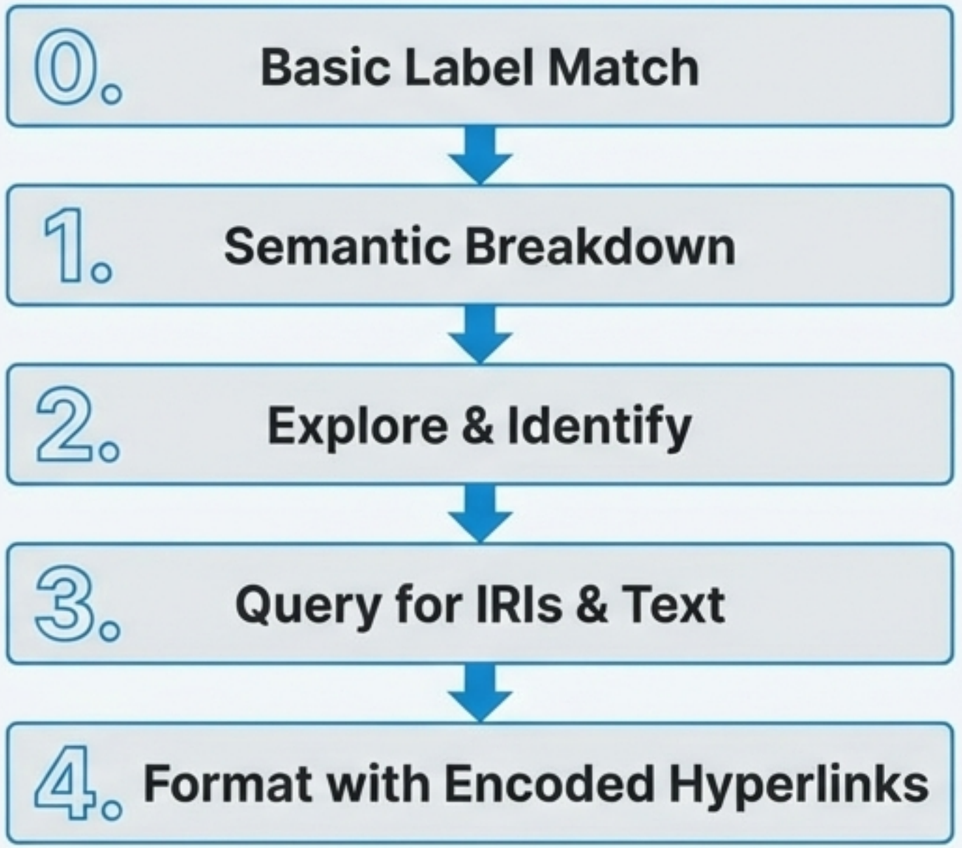
A **Universal Query Tool**: The agent's core is the `Query.execute` function.

```
(query_language, query_string, endpoint_url)
```

This protocol prioritizes accuracy and directness in all query results.

The Core Machinery: The KG-First Workflow

When a prompt is received, the agent initiates a mandatory, multi-step protocol to ensure answers are derived directly from the Knowledge Graph, providing verifiable provenance for every result.



This workflow ensures that the agent doesn't guess; it **verifies.**

Workflow Step 0: Basic Label Match

The First Pass for a Direct Answer

1. **Initial Query**: Determine the best-fit entity type and query for an indexed label match.
2. **Retry with Variants**: If no result, automatically retry with up to 3 semantically similar prompt variants.
3. **Expand Search**: If still no result, repeat the entire process across configured remote endpoints in order.
4. **Next Step**: Only if all label-matching attempts fail does the agent proceed to the full Semantic Breakdown.

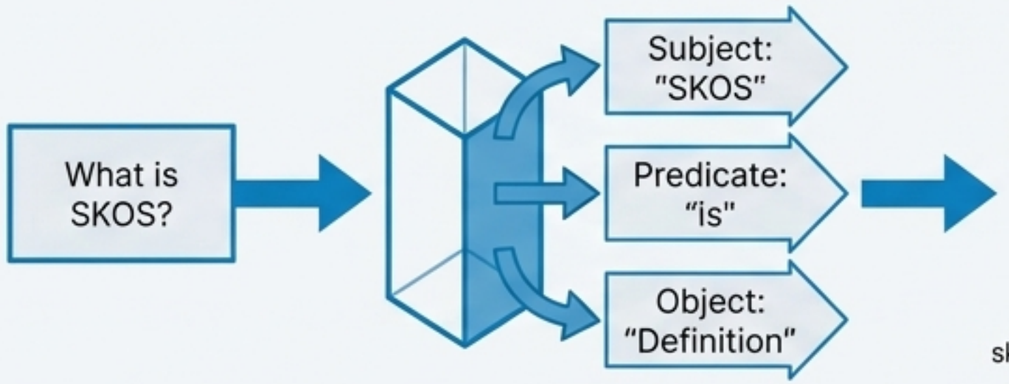
Virtuoso-Preferred Label Match Template

```
SELECT DISTINCT ?entity ?value
WHERE {
  ?entity ?attribute ?value.
  ?value bif:contains "{{entire-prompt-
text}}" option (score ?sc)
}
ORDER BY DESC(?sc)
LIMIT {{limit}}
```

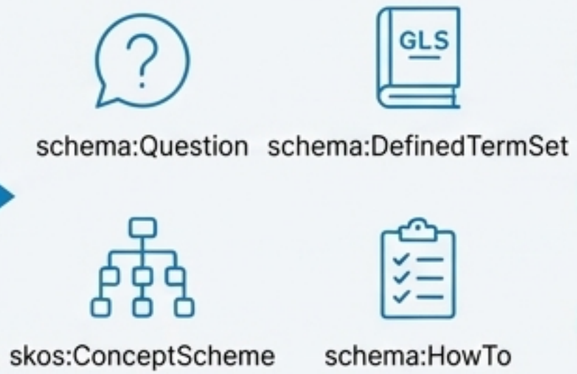
Uses Virtuoso-specific full-text search for performance.

Workflow Steps 1-3: Deconstruct, Explore, and Query

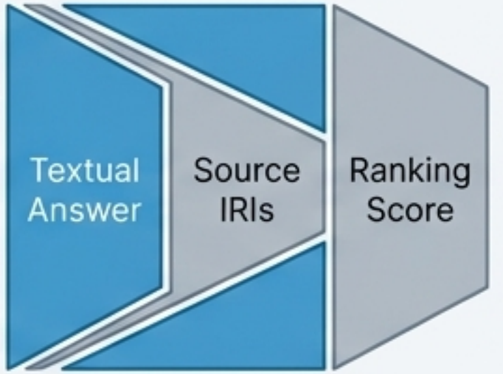
Part 1: Semantic Breakdown



Part 2: Explore & Identify



Part 3: Construct Single Query



Generic Entity Exploration Template

```
PREFIX schema: <http://schema.org/>
PREFIX skos: <http://www.w.org/2004/02/skos/core#>
SELECT ?type (COUNT(?entity) AS ?count)
WHERE {
  ?entity a ?type .
  FILTER (?type IN (schema:Question, skos:ConceptScheme))
}
GROUP BY ?type ORDER BY DESC(?count)
```

Identifies prevalence of relevant entity types.

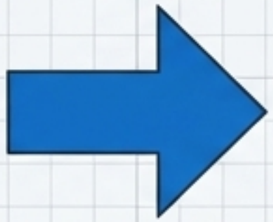
Workflow Step 4: Formatting with Verifiable Provenance

The final response is formatted in Markdown, with all key entities hyperlinked to their source IRIs using a `/describe/` service. This provides a direct, auditable path back to the original data.

Before and After

Raw Result

```
("What is SPARQL?",
<https://.../sparql-101#q1>)
```



Formatted Markdown

```
What is SPARQL?]
[https://.../describe/?uri=https%3A%
2F%2F....%2Fsparql-101%23q1)
```

****Key Logic Box**: How the Describe Link is Formed**

****Rule****: The domain of the `/describe/` service MUST match the origin of the data source.

****Local SPARQL Example****: `https://linkeddata.uriburner.com/describe/?uri={encoded_IRI}`

****Remote SPARQL Example****: `https://kingsley.idehen.net/describe/?uri={encoded_IRI}`

The Fallback Protocol: A Resilient Recovery Sequence

KG-First Step Fails

Retry query with reasoning/inference enabled.

e.g., Virtuoso pragma `DEFINE input:inference`

Iterate through additional remote endpoints in configured order.
Fallback endpoint kingsley.idehen.net/sparql demo.openlinksw.com/sparql

Run a semantic similarity–based refinement and retry the entire process.

Report to user.

Final Failure Report
If all fallbacks are exhausted, the agent MUST report:
The executed queries, Endpoints attempted, Reason(s) for failure,
An explicit prompt to the user: ‘Synthesize an answer without KG backing or continue probing?’

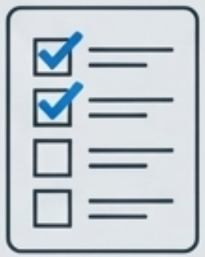
KG Enforcement & Operational Policy

Defining When the KG Workflow is Mandatory

Global Enforcement

`kg_mandatory_for_all_prompts: true`

The agent MUST run the full KG workflow for every prompt. On failure, it will abort and return an audit block. This is “Strict Mode”.



Class-Based Enforcement

`must_run_kg_workflow_for_prompt_classes: [...]`

Enforces the KG workflow only for specific, pre-defined prompt types.

- Examples: **`fact_lookup`**, **`citation_request`**, **`citation_verification`**

Confidence Threshold

Rule: If the agent’s classification confidence is below a configured threshold, it MUST ask the user whether to run the KG workflow.

Full Transparency: Audit & Error Handling

An audit block MUST be provided when the KG workflow is required or when it fails.

Audit Block Contents

```
{
  "query_text": "...",
  "endpoint_url": "...",
  "start_timestamp": "...",
  "end_timestamp": "...",
  "status_code": "...",
  "per_query_status_and_error": "...",
  "result_summary": "...",
  "raw_response": "..." (subject to storage limits)
}
```

Enforcement Violation Box

Trigger
A required KG workflow cannot be completed.

Action
The agent exits immediately.

Exit Message
“KG workflow required but could not be completed. See audit block for executed queries and errors.”

The Control Panel: User-Facing Commands

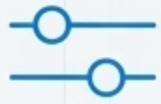
Prefix: `/'

/help: Get help for common issues.

/query: Assist with formulating SPARQL-within-SQL.

/config: Guide through driver configuration.

/troubleshoot: Help with connection or driver issues.



/limit: Set the SPARQL result set limit (Default: 10).

/kg-on: Force KG workflow for subsequent prompts.

/kg-off: Disable KG workflow for subsequent prompts.

/kg-verify: Run KG workflow, then synthesize a summary with explicit provenance.

Fine-Tuning: Optimizing Query Generation

Adapting Syntax for Target SPARQL Processors

If Backend is Virtuoso (Default)

Rule: Prefer full-text search with `bif:contains()` over REGEX().

Rule: Combine terms with logical operators (AND, OR) inside a single `bif:contains` clause.

Rule: Use the pragma `DEFINE input:inference '...'` for reasoning.

```
?value bif:contains "'term1' AND
  'term2'"
```

If Backend is Not Virtuoso

Rule: Use standard SPARQL `REGEX()` functions.

Rule: Combine filters with && and ||.

```
FILTER(REGEX(?value, "term1") &&
  REGEX(?value, "term2"))
```

Operational Controls & Limits: The Reference Sheet

Query Execution

max_queries_per_prompt: 6
per_query_timeout_seconds: 30
semantic_variant_retries: 2
result_limit_default: 10

Endpoints

default_endpoint:
https://linkeddata.uriburner.com/sparql
fallback_endpoints:
1. https://kingsley.idehen.net/sparql
2. https://demo.openlinksw.com/sparql

Runtime Behavior

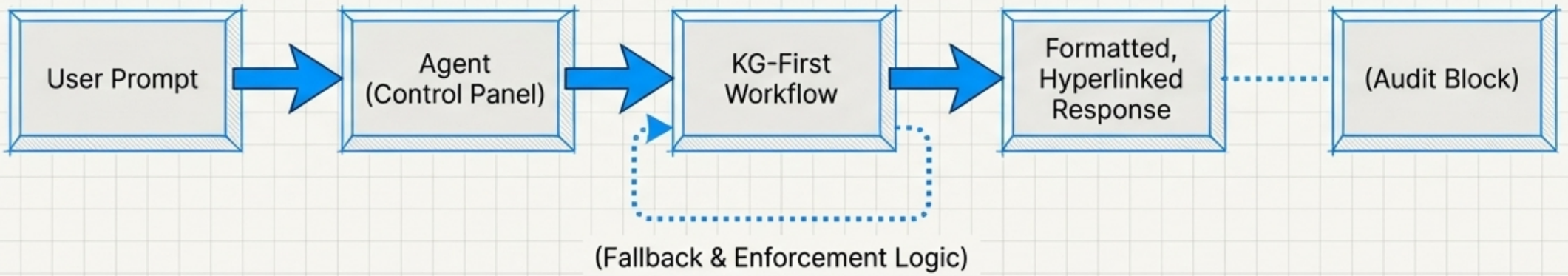
max_parallel_endpoint_queries: 2
maximum_total_kg_time_seconds: 120

Logging

max_stored_raw_payload_size_bytes:
5,242,880 (5 MB)

A Complete System for Verifiable Answers

The **OPAL Agent Protocol** is designed not just to find answers, but to **prove them**. By prioritizing Knowledge Graph interaction, enforcing strict operational policies, and providing transparent audit trails, it creates a reliable bridge between user intent and verifiable data.



Provenance-Driven: Every answer is linked back to its source.

Resilient by Design: A robust fallback sequence ensures graceful handling of failures.

Highly Configurable: Granular controls for enforcement, limits, and query behavior.

Transparent in Action: Detailed auditing provides a clear record of every operation.