Technical Design Specification: CognitiveGraph Semantic Schema Extension

1.1. Introduction

This document specifies the required extensions to the CognitiveGraph schema. The primary objective of these changes is to elevate the graph's representative power from a purely syntactic and structural model to a rich semantic knowledge base. By explicitly modeling high-level programming concepts such as data models and architectural patterns, this extended schema will provide the necessary foundation for the Golem project's advanced reasoning and code transformation capabilities.

1.2. Guiding Principles

- **Explicitness over Inference:** Abstract concepts that are critical for transformation (e.g., "this class is a data entity") should be made first-class citizens of the graph schema rather than being left as latent patterns for a neural network to infer.
- **Extensibility:** The schema must be designed for evolution. As new frameworks, languages, and architectural patterns emerge, it should be straightforward to extend the schema with new node and edge types to represent them.[1]
- **Interoperability:** The schema should be serializable into standard, efficient formats to facilitate communication between the Minotaur analysis platform and the Golem reasoning engine.

1.3. Proposed Schema Extensions

The existing CognitiveGraph schema, assumed to be a variant of a Code Property Graph (CPG), will be extended with new node and edge types designed to capture application-level semantics.

1.3.1. New Semantic Node Types

Two new high-level node types are introduced:

Node Type	Description	Attributes (Properties)	Example

DataModel	Represents a core data entity of the application. This is a higher-level abstraction than a simple Class or Struct.	name: string (e.g., "User") framework: string (e.g., "JPA", "Pydantic", "SQLAlchemy") stereotype: string (e.g., "Entity", "DTO", "ValueObject")	A class annotated with @Entity in Java Spring or a class inheriting from BaseModel in Pydantic.
ArchitecturalPattern	Represents an identified design or architectural pattern instance. This node anchors an abstract concept to a concrete set of code elements.	name: string (e.g., "UserController") patternType: string (e.g., "Controller", "Service", "Repository", "MessageHandler") framework: string (e.g., "Spring", "FastAPI")	A class annotated with @RestController in Spring or a function decorated with @app.get in FastAPI.

1.3.2. New Semantic Edge Types

Two new edge types are introduced to link the new semantic nodes to the existing structural graph:

Edge Type	Source Node(s)	Target Node(s)	Description
PART_OF_PATTERN	Class, Function, Module	ArchitecturalPattern	Connects a concrete code element to the abstract pattern it helps implement.
USES_DATA_MODEL	Function, Method, Class	DataModel	Creates a direct semantic link from a piece of logic to a data entity it operates on (creates, reads, updates, or deletes).

1.4. Graph Serialization and Exchange Format

To facilitate the transfer of the entire, potentially massive, CognitiveGraph from Minotaur to Golem, a high-performance, schema-aware binary serialization format is required. While

human-readable formats like JSON-LD or GraphML are useful for debugging, they are inefficient for large-scale data transfer.[3]
The recommended format is **Protocol Buffers (Protobuf)**.

Format	Pros	Cons	Suitability
Protocol Buffers	Extremely compact binary format, fast serialization/deserialization, strong typing and schema enforcement, excellent backward/forward compatibility.[5]	Not human-readable, requires a schema definition and compilation step.[6]	**High.** Ideal for efficient, type-safe, large-scale data exchange between services.
JSON-LD / GraphSON	Human-readable, excellent web and tooling support, flexible schema.[4]	Verbose, larger file sizes, slower parsing, lacks built-in strong typing.[3]	**Low.** Suitable for small graphs, debugging, or web-based visualization, but not for transferring project-scale graphs.
GraphML	Standardized XML format for graphs, supports attributes and nested graphs.[9]	XML verbosity leads to large file sizes, can be slow to parse.[3]	**Medium.** Better than plain JSON for graph structure but less performant than binary formats.

A .proto file will define the complete CognitiveGraph schema, including all node and edge types and their properties. This allows for the generation of highly optimized serialization and deserialization code in the native languages of both Minotaur and Golem.