

# VectorOWL + MCP: A Neuro-Symbolic Architecture for AI-Native Systems Engineering

---

Matthew Collaro

April 1, 2026

## Abstract

---

The increasing complexity of modern aerospace and automotive systems has exposed the structural limitations of traditional Model-Based Systems Engineering (MBSE) frameworks. Current symbolic modeling paradigms, such as SysML and OWL, lack the representational capacity to integrate high-dimensional vector data from simulations and real-world telemetry. This paper proposes **VectorOWL + MCP**, a novel neuro-symbolic architecture that extends the Web Ontology Language (OWL) with native vector embeddings and utilizes the Model Context Protocol (MCP) as a distributed runtime for model synchronization. We define a hybrid reasoning engine that combines description logic with manifold-based similarity operations, anchored by a deterministic constraint layer to ensure safety-critical correctness. This architecture facilitates a transition from static, model-driven engineering to a continuous, feedback-driven computational substrate.

## 1. Introduction and Problem Statement

---

Systems engineering for complex, safety-critical domains is undergoing a paradigm shift toward data-driven methodologies [1]. However, the foundational tools of the field—Model-Based Systems Engineering (MBSE) and its primary language, SysML—remain rooted in a purely symbolic tradition.

### 1.1 Limitations of Symbolic MBSE

Traditional MBSE systems are built upon discrete, first-order logic or description logic (DL). While these frameworks provide rigorous traceability and structural consistency, they suffer from several “epistemic gaps”:

- **Rigidity of Boolean Truth:** Symbolic systems operate on binary assertions, making them ill-equipped to handle the probabilistic and noisy nature of real-world sensor data and simulation outputs [2].
- **High-Dimensional Data Exclusion:** Modern engineering artifacts, such as Computational Fluid Dynamics (CFD) results and Finite Element Analysis (FEA) meshes, are naturally represented as high-dimensional vectors. Symbolic ontologies cannot natively “reason” over these representations without significant information loss during abstraction.
- **Latency in Synchronization:** The lack of a unified runtime protocol leads to fragmented engineering lifecycles, where design models, simulation results, and telemetry data exist in disconnected silos.

### 1.2 The Need for Neuro-Symbolic Integration

To address these challenges, we require a “hybrid reasoning” approach that preserves the logical rigor of symbolic models while leveraging the statistical power of learned representations [3]. This paper introduces VectorOWL as the structural foundation and MCP as the operational protocol for this new paradigm.

## 2. VectorOWL: Manifold-Augmented Ontologies

---

VectorOWL extends the standard OWL 2 specification by mapping ontological entities (classes, individuals, and properties) into a continuous vector space  $\mathcal{V} \subset \mathbb{R}^d$ .

## 2.1 Native Vector Embeddings and Entity Data Structures

In VectorOWL, every entity  $e$  is uniquely identified by a Universal Resource Identifier (URI) and encapsulates both symbolic and statistical representations. The core `Entity` data structure can be formally defined using a Protocol Buffer-like schema for efficient serialization and deserialization across distributed systems:

```
message Entity {
  string uri = 1; // Unique identifier for the entity (e.g., "http://example.com/AircraftA")
  enum EntityType {
    CLASS = 0;
    INDIVIDUAL = 1;
    PROPERTY = 2;
  }
  EntityType type = 2;
  AxiomSet axioms = 3; // Symbolic axioms associated with the entity
  EmbeddingManifold embedding = 4; // High-dimensional vector representation
  map<string, string> attributes = 5; // Key-value store for discrete attributes
}

message AxiomSet {
  repeated Triple triples = 1;
}

message Triple {
  string subject_uri = 1;
  string predicate_uri = 2;
  oneof object_type {
    string object_uri = 3;
    string literal_value = 4;
  }
}

message EmbeddingManifold {
  repeated float vector = 1 [packed=true]; // The d-dimensional vector embedding (e.g., Float32Array)
  repeated EmbeddingSource source_metadata = 2;
}

message EmbeddingSource {
  string source_id = 1; // Identifier for the data source (e.g., "CFD_Sim_123")
  string timestamp = 2; // ISO 8601 format
  string algorithm = 3; // Algorithm used for embedding (e.g., "Word2Vec", "BERT", "CustomCNN")
  map<string, string> parameters = 4; // Algorithm-specific parameters
}
```

Each entity  $e$  is thus a composite data structure, formally represented as:  $\$e = \langle \text{uri}, \text{type}, \mathcal{S}, \mathbf{v}, \mathcal{A} \rangle$  where  $\mathcal{S}$  is an *AxiomSet* representing symbolic assertions (e.g.,  $e \sqsubseteq C$ ),  $\mathbf{v} \in \mathbb{R}^d$  is the *EmbeddingManifold* vector learned from multi-modal data sources (e.g., *CAD geometry*, *performance metrics*, and *historical telemetry*), and  $\mathcal{A}$  denotes the `attributes` map for discrete properties.

## 2.2 Hybrid Reasoning Mechanism

The reasoning engine in VectorOWL operates across two interconnected layers, facilitating a neuro-symbolic inference capability:

- 1. Symbolic Layer (Graph):** This layer processes the `AxiomSet` of entities. It employs a tableaux-based algorithm, such as those found in highly optimized OWL reasoners [5], to ensure logical consistency, satisfiability, and infer new symbolic relationships. This ensures adherence to formal ontological constraints.
- 2. Statistical Layer (Vector):** This layer operates on the `EmbeddingManifold` vector of entities. For two entities  $e_1, e_2$ , the relationship strength is computed via a kernel function  $K(\mathbf{v}_1, \mathbf{v}_2)$ , typically a cosine similarity or a learned distance metric in a hyperbolic space to better represent hierarchical structures [6]. This allows for probabilistic inference based on semantic similarity.

The hybrid inference rule can be formalized as:  $\$Inference(e_1, e_2) = \alpha \cdot 1(\mathcal{S}_1 \vdash \mathcal{S}_2) + (1 - \alpha) \cdot K(\mathbf{v}_1, \mathbf{v}_2)$  where  $\alpha \in [0, 1]$  is a weighting factor that determines the degree of symbolic versus statistical influence. The optimal  $\alpha$

*canbelearnedviameta –*

*optimizationonavalidationdataset, balancinglogicalprecisionwithstatisticalgeneralization. Thisoptimizationcanbeframed as  $L(\alpha)$  over a set of ground – truth inferences, where  $L$  combines errors from both symbolic and statistical predictions.*

### 3. MCP: The Engineering Runtime Kernel

---

The Model Context Protocol (MCP) serves as the “operating system” for the VectorOWL architecture, providing a standardized interface for tool interoperability and real-time synchronization [7].

#### 3.1 Distributed Identity and Context Management

MCP maintains a persistent, URI-based identity layer across heterogeneous engineering environments. By implementing a “Context Server” at each tool node (e.g., CATIA, Ansys, MATLAB), MCP ensures that changes in a local tool’s state are immediately propagated to the global VectorOWL graph. The `IdentityRegistry` is a crucial data structure for this, mapping local tool-specific identifiers to global VectorOWL URIs:

```
message IdentityRegistryEntry {
  string local_tool_id = 1; // Identifier from the local tool (e.g., CAD part number)
  string vectorowl_uri = 2; // Corresponding global VectorOWL URI
  string tool_type = 3; // Type of the originating tool (e.g., "CAD", "FEA", "PLM")
}

message IdentityRegistry {
  repeated IdentityRegistryEntry entries = 1;
}
```

This registry is distributed and eventually consistent, managed by a consensus protocol among MCP Context Servers.

#### 3.2 Event-Driven Synchronization and Dependency Tracking

The MCP runtime utilizes an asynchronous, event-driven architecture. We model the engineering system as a directed acyclic graph (DAG) of dependencies, where nodes are `Entity` URIs and edges represent functional or structural dependencies. When an upstream design parameter  $p$  is modified, MCP triggers a `ContextUpdate` event. The schema for this event, using Protocol Buffers, is designed for high-throughput, low-latency communication:

```
message ContextUpdate {
  string target_uri = 1; // URI of the entity being updated
  enum ActionType {
    CREATE = 0;
    UPDATE = 1;
    DELETE = 2;
  }
  ActionType action = 2;
  bytes payload = 3; // Serialized data representing the change (e.g., new vector, axiom diff)
  string timestamp = 4; // ISO 8601 format
  string source_tool_id = 5; // Identifier of the tool originating the update
  string transaction_id = 6; // Unique ID for a series of related updates
}
```

This event propagates through the DAG, triggering re-evaluations of downstream simulations and anchor constraints in real-time. The dependency graph itself is maintained as an adjacency list or matrix, allowing for efficient traversal and topological sorting for update propagation. The `vectorowl` daemon (Section 5.2) processes these events using a work-stealing queue and a thread pool, ensuring minimal latency.

### 4. Anchors: Deterministic Enforcement in a Probabilistic World

---

A primary risk in neuro-symbolic systems is the potential for “hallucinations” or unsafe inferences in the statistical layer. To mitigate this, we introduce **Anchors** as a deterministic enforcement layer.

## 4.1 Formal Definition of Anchors and Data Structures

An anchor  $\mathcal{A}$  is a hard constraint that must be satisfied for a design to be considered valid. Anchors are defined as predicates over the system state:  $\mathcal{A} : \Sigma \rightarrow \{0, 1\}$  where  $\Sigma$  is the set of all symbolic and vector attributes. Each anchor is an executable object with a defined evaluation function and associated metadata. The `Anchor` data structure is defined as:

```
message Anchor {
  string uri = 1; // Unique identifier for the anchor
  string target_entity_uri = 2; // The URI of the entity this anchor constrains
  enum AnchorType {
    SCALAR = 0;
    RELATIONAL = 1;
    FUNCTIONAL = 2;
  }
  AnchorType type = 3;
  string predicate_code = 4; // Executable code (e.g., WebAssembly, Python bytecode) for the predicate function
  map<string, string> parameters = 5; // Parameters for the predicate (e.g., threshold value, reference URI)
  enum Severity {
    WARNING = 0;
    ERROR = 1;
    CRITICAL = 2;
  }
  Severity severity = 6;
  string last_evaluated_timestamp = 7; // ISO 8601 format
  bool is_satisfied = 8;
  string evaluation_log = 9; // Detailed log of the last evaluation
}

message AnchorRegistry {
  map<string, Anchor> anchors = 1; // Map from anchor URI to Anchor object
}
```

## 4.2 Anchor Taxonomy

Anchor Type	Mathematical Form	Engineering Example
Scalar	$f(x) \leq \theta$	Operating temperature < 150°C
Relational	$x \in \text{Neighbors}(y)$	Component $A$ must be physically mated to $B$
Functional	$y = g(x_1, \dots, x_n)$	Lift-to-drag ratio computed via Navier-Stokes

Anchors act as the “ground truth” layer, overriding any probabilistic suggestions from the vector layer if a violation is detected. This ensures that the system remains within the bounds of safety-critical correctness [8]. The `AnchorRegistry` is continuously monitored by a dedicated constraint solver, which can be implemented using Satisfiability Modulo Theories (SMT) solvers for complex logical constraints or custom rule engines for performance-critical functional constraints.

## 5. System Architecture and Implementation

The proposed architecture is implemented as a multi-layered stack, optimized for low-latency execution on Linux-based high-performance computing (HPC) environments.

### 5.1 Layered Stack Model

- Ontology Layer (OWL/RDF):** Stores the structural and logical definitions using a graph database (e.g., Neo4j, RDF triple store). This layer primarily manages `Entity.axios` and supports SPARQL-like queries for symbolic inference.
- Vector Layer (HNSW/VectorDB):** Manages high-dimensional embeddings and similarity queries using an Approximate Nearest Neighbor (ANN) index (e.g., HNSW, Faiss) for `Entity.embedding.vector`. This layer is optimized for fast vector search and updates, often residing in GPU memory for accelerated computations.
- Anchor Layer (Constraint Solver):** Executes deterministic validation checks. The `AnchorRegistry` is continuously monitored by a dedicated constraint solver, which can be implemented using SMT solvers or custom rule engines,

potentially leveraging formal verification techniques.

- MCP Layer (Runtime):** Handles tool-to-model communication and event orchestration. This layer manages the `IdentityRegistry` and processes `ContextUpdate` events, acting as the central nervous system for the entire engineering ecosystem.

## 5.2 Implementation Strategy: `vectorowl`

The core runtime, `vectorowl`, is implemented in **Rust** to leverage its memory safety, zero-cost abstractions, and robust concurrency primitives. It utilizes `io_uring` for high-throughput asynchronous I/O operations, crucial for handling rapid telemetry streams and simulation outputs without blocking. Memory-mapped files (`mmap`) are extensively used for the `EmbeddingManifold` and `AxiomSet` to minimize data transfer overhead between disk and RAM, enabling near real-time access to large-scale models. Concurrency is managed through a combination of lock-free data structures (e.g., RCU for read-heavy operations on the graph) and message-passing paradigms (e.g., Tokio channels) for inter-component communication within `vectorowl`. The daemon exposes a gRPC API for external MCP Context Servers to interact with the VectorOWL graph and trigger updates.

## 6. Comparative Analysis

Feature	SysML v2	Digital Twin	VectorOWL + MCP
Reasoning Mode	Symbolic/Logical	Physics-based	Hybrid (Neuro-Symbolic)
Data Integration	Manual/API-based	Real-time Stream	Real-time Manifold Sync
Safety Layer	Static Constraints	Simulation-based	Dynamic Deterministic Anchors
AI Readiness	Low	Medium	High (Native Embeddings)

## 7. Use Cases and Industrial Application

### 7.1 Aerospace: Aircraft Design Optimization

In aircraft design, VectorOWL enables “semantic design reuse” by identifying past wing configurations that are statistically similar in performance to a new requirement set, while the anchor layer ensures that all proposed designs meet FAA structural safety margins. This reduces design cycle time and improves design quality by leveraging historical data.

### 7.2 Automotive: Closed-Loop Failure Detection

By embedding real-time telemetry from vehicle fleets into the VectorOWL space, manufacturers can detect anomalies that “cluster” near known failure modes in the vector space, triggering immediate MCP-based alerts to the design engineering team for root-cause analysis. This proactive approach minimizes downtime and enhances vehicle safety.

## 8. Vision: The Feedback-Driven Engineering Substrate

The transition from model-driven to **feedback-driven** engineering represents the final stage of industrial digitalization. In this vision, the engineering system is no longer a static blueprint but a continuous, learning organism that evolves as new data is ingested. VectorOWL + MCP provides the unified computational substrate necessary to realize this future.

## 9. Conclusion

The complexity of modern systems engineering demands a departure from purely symbolic modeling. VectorOWL + MCP offers a rigorous, neuro-symbolic framework that bridges the gap between abstract logic and high-dimensional data. By anchoring statistical learning in deterministic constraints, this architecture provides a safe and scalable foundation for the next generation of AI-native engineering.

## References

---

1. [1] INCOSE, “Systems Engineering Vision 2035,” International Council on Systems Engineering, 2022.
2. [2] Pearl, J., *The Book of Why: The New Science of Cause and Effect*, Basic Books, 2018.
3. [3] Garcez, A. d., & Lamb, L. C., “Neurosymbolic AI: The 3rd Wave,” *Artificial Intelligence Review*, 2023.
4. [4] Chen, J., et al., “OWL2Vec\*: Embedding of OWL Ontologies,” *Machine Learning*, 2021.
5. [5] Horrocks, I., “The FaCT System,” *Tableau Conference*, 1998.
6. [6] Nickel, M., & Kiela, D., “Poincaré Embeddings for Learning Hierarchical Representations,” *NeurIPS*, 2017.
7. [7] Anthropic, “Model Context Protocol (MCP) Specification,” 2024. [Online]. Available: <https://modelcontextprotocol.io>
8. [8] Leveson, N. G., *Engineering a Safer World: Systems Thinking Applied to Safety*, MIT Press, 2011.