

Unified Network Abstraction Layer via Knowledge Graph

Felipe Thomas, Eduardo Canto, Vinicius
Guimarães, Julião Braga, **Jéferson Nobre**

Summary

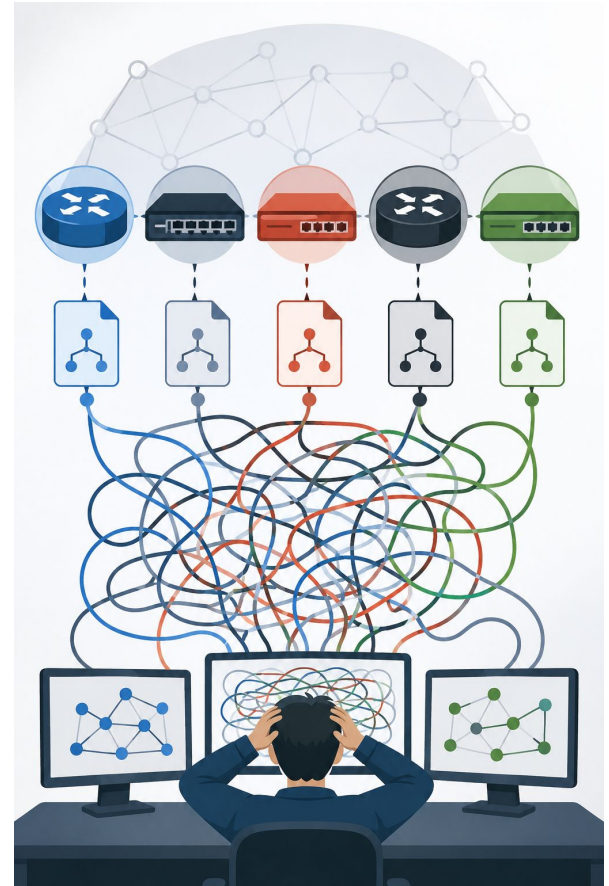
- Motivation
- Related Work
- Proposed Solution
- Three-Layer Information Model
- Results
- Conclusions

The Heterogeneity Problem

YANG is the standard modeling language for network configuration and states (NETCONF/RESTCONF).

The Issue: High structural and semantic fragmentation across vendors.

IAB NEMOPS 2024 Report (RFC 9968) highlighted that distinct models hinder network automation.



The Multivendor Challenge

A Minimal Case: The admin-status attribute. Even simple configurations are represented differently across vendors:

- **Cisco (IOS-XR):** `/interface-configurations/.../shutdown` (*empty type*)*
 - **Huawei:** `/ifm:interfaces/.../admin-status` (*enumeration: up/down*)
 - **Juniper:** `/interfaces/.../disable` (*empty type*)*
- * In Cisco and Juniper, an empty leaf means the interface is administratively down.

One concept, **three completely different implementations**

This heterogeneity is repeated across hundreds of YANG modules

Related Work

Approach	Key Authors	Main Limitation / Gap
Structural Conversion (YANG → RDF mapping)	Mackey et al. (2024) Martínez et al. (2025)	RDF transformation only; preserves vendor-specific semantics.
Analytics & Operations (Anomaly detection, traffic)	CANDIL (2024) NORIA (2023) Pang et al. (2025)	Highly tailored to specific tasks; limited cross-vendor semantic inference.
Unified Abstraction (Multi-vendor semantic abstraction)	This Work	Harmonizes multi-vendor YANG models for unified semantic queries.

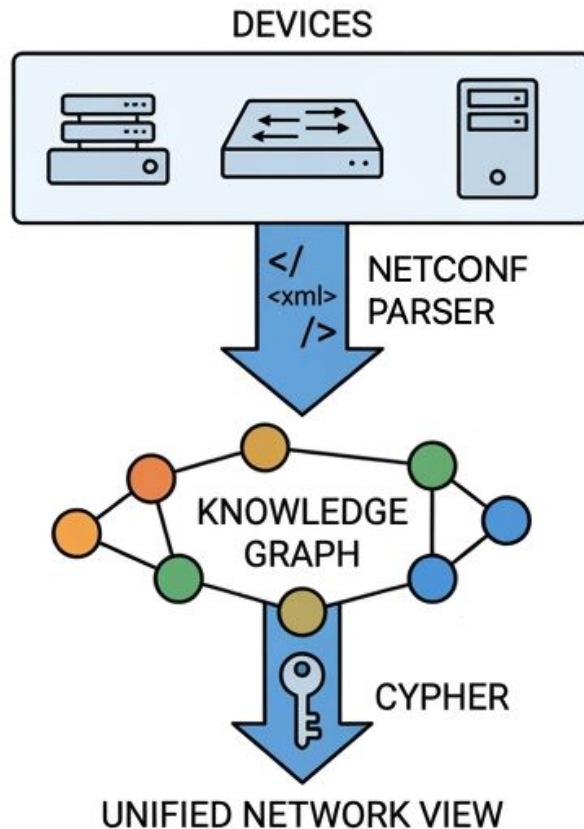
Contributions

This work makes three main contributions:

- Introduces a vendor-independent semantic model capable of representing common network concepts despite differences among YANG models;
- Proposes a three-layer Knowledge Graph that separates semantic concepts, vendor mappings, and operational data while preserving the original vendor-specific structures;
- Finally, the proposed abstraction enables vendor-independent semantic queries.

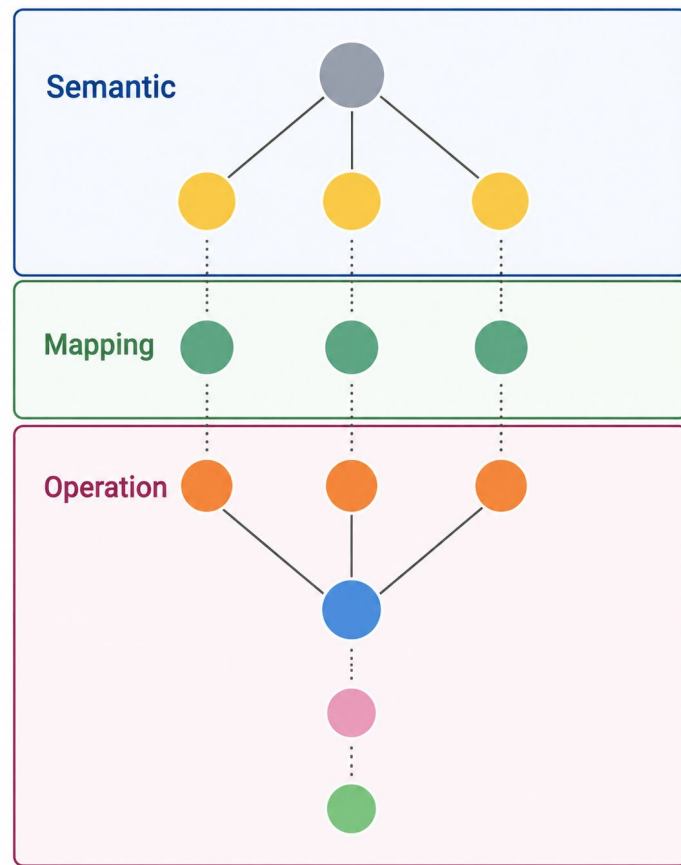
Proposed Architecture

Our architecture places a Knowledge Graph (Neo4j) between network devices and management applications. Data collected from heterogeneous YANG models via NETCONF is integrated into a unified semantic representation, allowing applications to perform vendor-independent queries while preserving the original vendor-specific models.



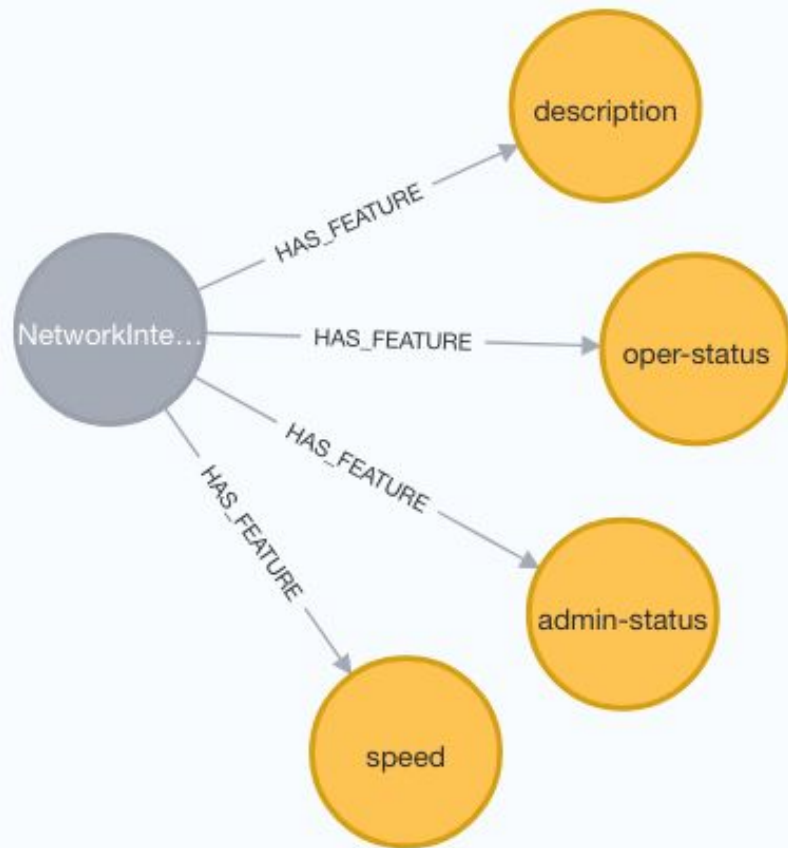
Three-Layer Information Model

Network management data is organized into three complementary layers: a semantic layer that defines vendor-independent concepts, a mapping layer that relates these concepts to vendor-specific YANG implementations, and an operation layer that stores the actual network state. This organization enables vendor-independent queries while preserving the original YANG structures.



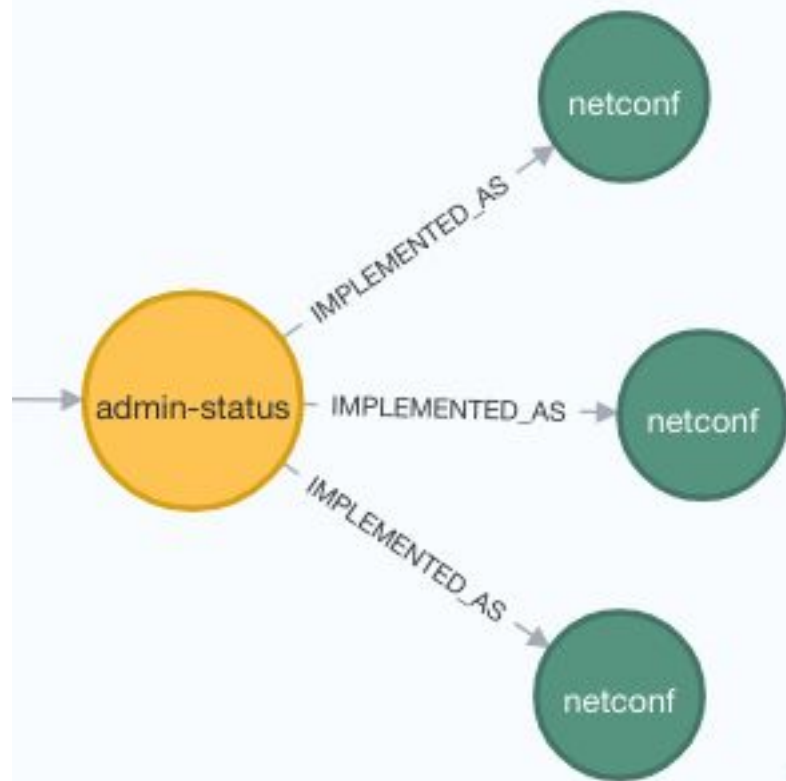
Semantic Layer

The semantic layer describes general concepts applicable to any network interface, regardless of the vendor. In this layer, the abstract node **InterfaceType** is established, representing the universal concept of a network interface.



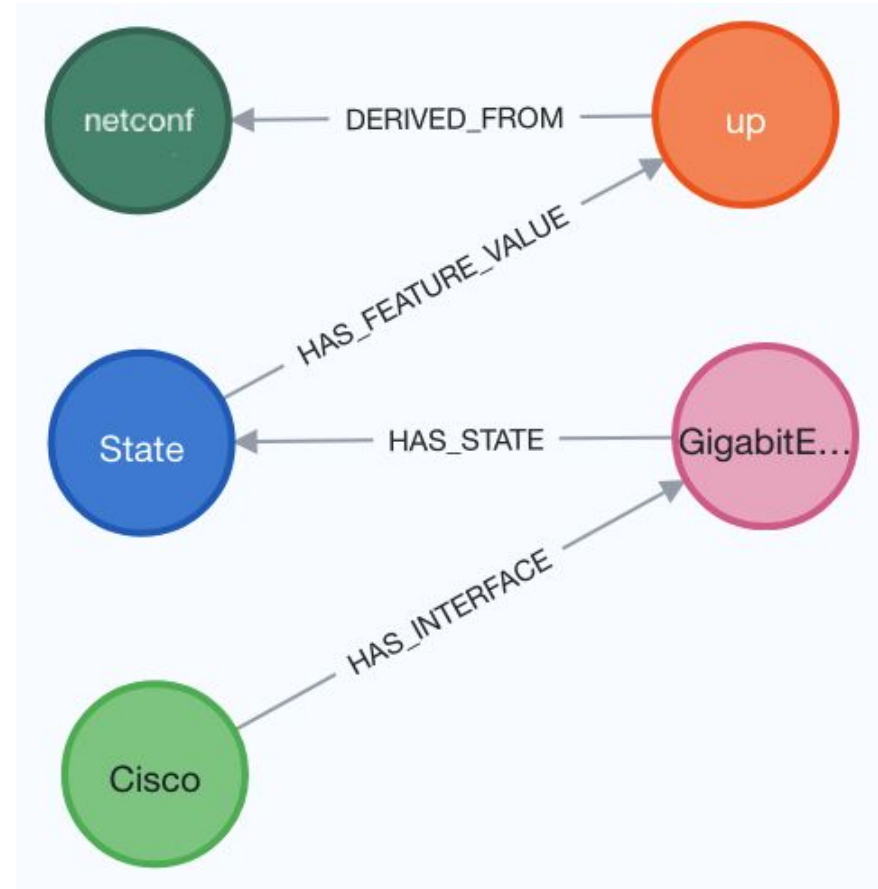
Multi-Vendor Mapping Layer

The mapping layer establishes the link between the semantic model and the vendor-specific representations. In this layer, each **Feature** in the semantic layer is associated with one or more nodes of type **FeatureImplementation**, which preserve the original structure modeled in each vendor's YANG modules.



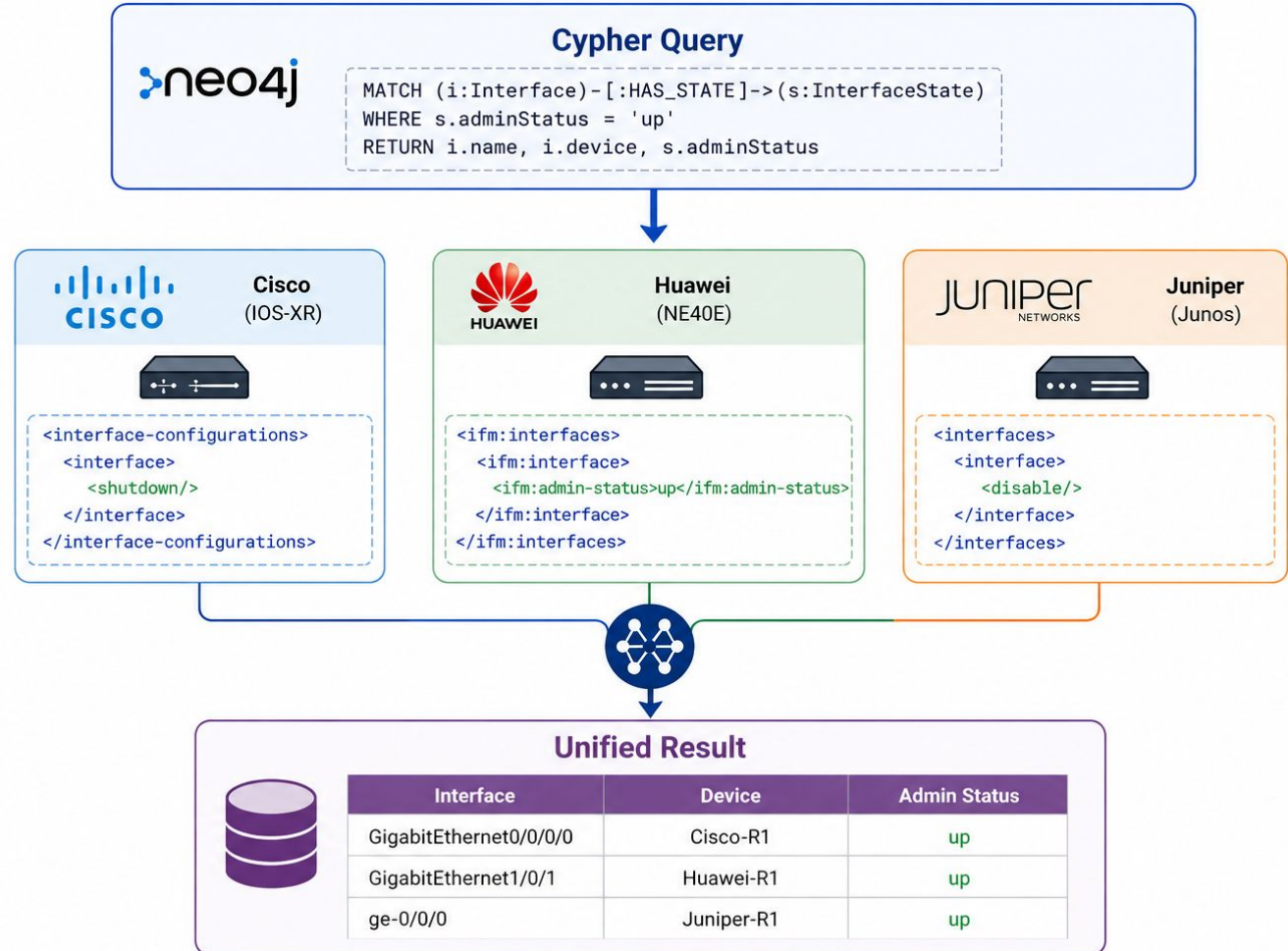
Operation Layer

The operation layer represents the operational state of the network, storing the values effectively observed on devices. It is in this layer that concrete instances of attributes appear, reflecting the configuration and operational behavior captured via NETCONF.



Results

- Single query
- Multiple vendors
- Unified Result



Results

The mapping layer automatically resolves YANG paths from semantic concepts, enabling dynamic NETCONF RPC generation across multiple vendors.

```
MATCH (:Feature {name: "admin-status"}) - [:IMPLEMENTED_AS]
->(fi:FeatureImplementation) WHERE fi.vendor = "Huawei"
RETURN fi.vendor, fi.yang_path;
```

Vendor	YANG Path
Huawei	/ifm:interfaces/ifm:interface/ifm:admin-status

Results

- **Single Point of Inference:** The semantic layer proved effective in defining a unified meaning, simplifying queries regardless of the underlying vendor.
- **Automated Discovery:** The graph resolves vendor-specific YANG paths, enabling systems to dynamically construct NETCONF RPC payloads.
- **Unified State Retrieval:** A single Cypher query successfully retrieves all administratively active interfaces simultaneously across Cisco, Huawei, and Juniper devices.
- **Contextual Semantics:** By bridging operational data with the semantic layer, raw network values gain rich, standardized contextual meaning.

Conclusions

- Unified semantic abstraction for heterogeneous YANG models
- Three-layer information model preserving vendor-specific implementations
- Vendor-independent network queries through Knowledge Graphs
- Proof of concept validated across Cisco, Huawei, and Juniper

Future Research Directions

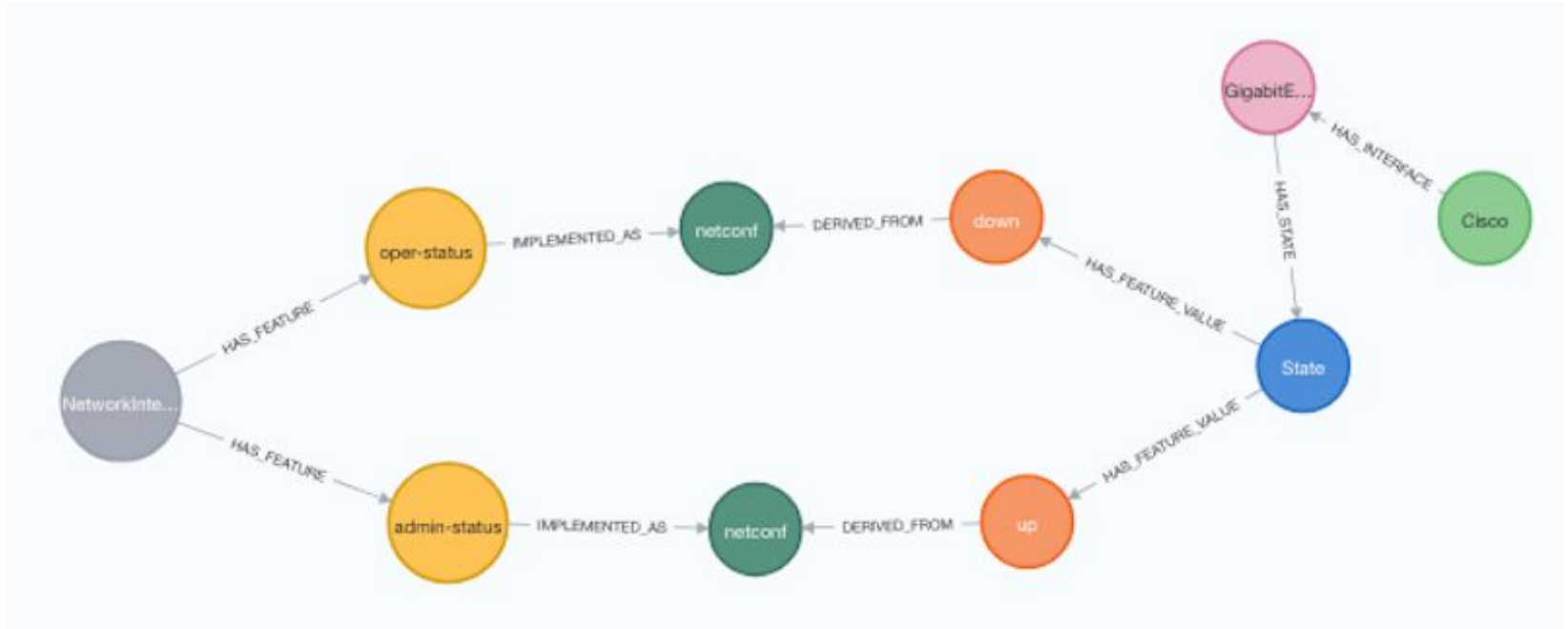
- Automation of ontology creation using Large Language Models (LLMs) for automatic parsing of YANG modules and generation of the mapping layer;
- Integration with real-time network monitoring tools; expansion of the ontology by implementing additional functionalities from the YANG models targeted in this work;

Questions?

Thank you for your attention

ANRW 2026

Full Graph



The Semantic Layer defines vendor-independent concepts (gray and yellow nodes on the left), the Multi-Vendor Mapping Layer links these concepts to vendor-specific YANG implementations (green in the middle), and the Operation Layer stores network instances and their current values retrieved via NETCONF (orange, blue, pink, and light green nodes).

Why Neo4j?

Why a Property Graph Model?

- **Native Graph Traversals**
 - Optimized for deep relationship searches (e.g., topology pathfinding) that are computationally expensive in relational databases.
- **Flexible Schema Evolution**
 - Allows adding new YANG modules or attributes (e.g., new **Feature** nodes) without destructive database migrations or downtime.
- **Efficient Semantic Relationships**
 - Direct mapping of semantic layers (Semantic -> Mapping -> Operation) using native relationships, enhancing query performance and readability.

Handling Network Complexity & Scale

How does the approach handle larger/more complex environments?

Our architecture is designed to be modular. The key point is that the **Semantic Layer** remains fixed, while the instances in the **Mapping** and **Operation layers** scale independently. Since we employ a graph-based approach, the complexity of adding a new vendor is linear: one simply needs to map the nodes and relationships of the new YANG module to our existing semantic layer. Furthermore, network topology is naturally represented as a graph, making dependency queries—such as 'what happens if this node fails?'—significantly more efficient than in tabular models.

Knowledge Graph vs. Time-Series Databases (TSDB)

Why not just use a Prometheus/InfluxDB for this?

A TSDB is excellent for storing raw time-series data, but it is completely unaware of schema semantics. The reason we used a Knowledge Graph—even for a minimal case like `admin-status`—is because the challenge isn't storing the value, but **interpreting** it across different vendors. A TSDB cannot inherently understand that a missing leaf in Cisco means 'Enabled', while an explicit enumeration in Huawei means 'Up'. By using a graph, we solve this semantic heterogeneity at the structural level. Once this foundation is proven for `admin-status`, it can be seamlessly combined with TSDBs in production: the TSDB handles high-frequency metric history, while our graph handles the semantic cross-vendor orchestration

NETCONF Parser

How does the parser handle vendor-specific YANG structures like empty leaves vs. enums?

The parser acts as a semantic translation layer. We don't just dump the raw XML value into the graph. For vendors like Cisco or Juniper, where **admin-status** is an **empty** leaf, the parser logic interprets the **presence or absence** of the leaf as a boolean/state mapping. It then instantiates the corresponding **FeatureValue** node—such as 'up' or 'down'—in the knowledge graph. This ensures that when an application queries the graph, it receives a normalized, vendor-agnostic status, regardless of whether the source device sent an XML **empty** tag or an **enumeration** string. We are effectively normalizing the operational intent, not just the data format.