NMOP
Internet-Draft
Intended status: Standards Track
Expires: 23 April 2026

W. Henderickx, Ed.
Nokia

20 October 2025

Graph based Meta Schema for collaborative Operations
draft-henderickx-meta-graph-schema-00

Abstract

Graph based Meta Schema for collaborative Operation

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 23 April 2026.

Table of Contents

1. Introduction

Modern automation environments require systems, users, and tools to operate on various data models in a coordinated manner. These models typically span heterogeneous technologies and administrative domains. Existing data modeling approaches, such as YANG, OpenAPI, and JSON Schema, ... focus primarily on syntax and validation but provide limited support for collaboration and conflict-aware operations.

The Graph-based Meta Schema introduces a graph-oriented representation of resources and their relationships, effectively forming a knowledge graph that connects data, intent, and operational context. This knowledge graph provides a unified semantic layer where machine and human actors can reason about dependencies, ownership, and intent across distributed systems.

By representing APIs and their relationships as a graph, the Meta Schema enables higher-level reasoning and automation patterns such as impact analysis, dependency tracing, and intent reconciliation. It provides a foundation for agents, controllers, and user interfaces to interact consistently, enabling multi-party automation where intent, observation, and orchestration operate through the meta schema.

This mechanism does not replace existing modeling languages; rather, it defines a meta-layer that integrates them and standardizes how resources and relations are described, linked, and managed.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.1. Terminology

* Resource:
  A distinct API object defined by the schema, representing an API modelled entity using YANG, openAPI, JSONSchema, ... A resource is identified using a unique identifier by group/version/kind. A resource could be global scoped or tenant scoped.
  - Group: A resource group is a logical grouping of resources. It disambiguates different APIs that may happen to have identically named kinds
  - Version: A resource version defines the stability of the API and backward compatibility guarantees
  - Kind A resource kind is the name of the API
  - Scope A resource scope defines if a resource is global or tenant scoped.
* Node:
  A graph vertex representing a resource instance with a unique instance api identifier using group/version/kind/name/(tenant)
* Edge:
  A directed relationship between two nodes (resource) that expresses containment, dependency, or association and the respective constraints.
* Meta Schema:
  A schema of schemas that describes how resources, attributes, relations, and operations are defined, versioned, and composed.
* Conflict Management:
  Mechanisms that allow multiple actors (human or automated) to operate on shared resources safely, including field ownership, merge rules, and version control.

3. Meta Schema

The Meta Schema provides a structural and semantic layer that integrates existing modeling frameworks (e.g., YANG, OpenAPI, JSON Schema) under a unified model. Implementations MAY embed or reference definitions from these sources as resources.

The Meta Schema defines:

1. Graph Topology how resources are linked via typed relations and its constraints.
2. Schema Semantics how attributes, lists, and maps are defined, including types, constraints and validation rules.
3. Operational Semantics how API operations (create, update, delete, apply, watch, etc.) interact with resource state.
4. User Semantics how metadata supports visualization, grouping, and human-friendly interaction (e.g., labels, icons, colors, categories).

This integration allows a single schema definition to be reused across API servers, UI frameworks, and automation agents/workflows.

4. Graph based schema

The graph model treats each resource as a Node and each relationship as an Edge. Relationships MUST specify direction and type, and MAY include additional attributes/constraints (e.g., cardinality, constraints, requiredness).

Example relations include:

* Parent/Child — hierarchical ownership
* RefersTo — loose association between resources
* Implements — specialization or binding to another kind
* DependsOn — operational dependency

Each node carries structured metadata such as labels, annotations, status, and version, allowing tools to reason about topology, impact, and dependency chains in a consistent way.

5. API operations

The Meta Schema defines conventions for API operations acting on resources. These conventions enable consistent behavior across REST, GraphQL, and streaming interfaces.

5.1. Conflict Management

Resources SHOULD support field-level ownership tracking (using the _managedFields_ in the metadata header) to allow concurrent edits by multiple actors. Conflicts MUST be detectable and MAY be resolved using merge policies.

5.2. Coordinated Deletion and Ownership

Resources MAY declare finalizers—named hooks that coordinate deletion until dependent resources have completed their cleanup. Ownership and relation semantics SHOULD guide cascading delete behavior.

5.3. Multi-Tenancy and Version Control

A resource definition MAY declare its tenancy scope (e.g., global, tenant). Version control metadata (e.g., generation, resourceVersion, branch identifiers) SHOULD be used to coordinate changes across distributed automation systems.

5.4. Bulk and Dry-Run Operations

Implementations SHOULD support bulk operations (e.g., batch delete) and dry-run modes to safely preview changes before applying them.

5.5. Streaming and Eventing

The schema MAY declare which resources are observable through event streams (e.g., watch, subscribe) and specify their event types. This enables synchronization between controllers, UI components, and external systems.

5.6. Opaque Metadata

Resources SHOULD support extensible metadata fields such as labels and annotations to carry user- or system-specific key/value pairs without affecting core semantics.

6. Example schema

```
apiVersion: apiextensions.c4o.dev/v1
kind: Schema
metadata:
    name: devices.assets.kuid.dev
spec:
    groups:
    - name: assets.kuid.dev
        apis:
        - resource: devices
          kind: Device
          scope: tenant
          ui:
            categories:
            - assets
            - devices
            displayName: Devices
            icon: xxx
            order: 100
        versions:
        - name: v1
          storage: true
          attributes:
          - # YANG, OpenAPI, JSONSchema, ...
          relationships:
          - name: parent
            target:
              group: location.kuid.dev
              version: v1
              kind: Site
            cardinality: one
          - name: parent
            target:
              group: assets.kuid.dev
              version: v1
              kind: Rack
            cardinality: one
```

7. IANA Considerations

This document has no IANA actions at this time.

8. Security Considerations

Schema definitions can expose sensitive structure and metadata about a system. Implementations MUST ensure that access control and data filtering mechanisms prevent unauthorized disclosure. Field ownership and event streaming MUST be authenticated and authorized according to the system’s security model.

9. References

* [RFC2119] S. Bradner, _Key words for use in RFCs to Indicate Requirement Levels_, March 1997.
* [RFC8174] B. Leiba, _Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words_, May 2017.

Acknowledgements

The author thanks colleagues and contributors from the network automation and kubenet community for discussions leading to this work.

Authors' Addresses

Wim Henderickx Nokia 50 Copernicuslaan 2018, Antwerp Belgium

Email: wim.henderickx@nokia.com