

NMOP
Internet-Draft
Intended status: Standards Track
Expires: 6 May 2026

W. Henderickx, Ed.
Nokia
2 November 2025

Graph based Meta Schema for collaborative Operations
draft-henderickx-meta-graph-schema-01

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 6 May 2026.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	2
2. Goals and Scope	3
3. Conventions and Definitions	3
4. Conceptual Model	3
5. Meta Schema	4
6. Graph based schema	5
7. API Operations	5
8. Meta Header	6
9. Example resource instance including meta header	6
10. Example schema	7
11. IANA Considerations	8
12. Security Considerations	9
13. References	9
Acknowledgements	9
Authors' Addresses	9

1. Introduction

Modern automation environments require systems, users, and agents to operate on shared data models in a coordinated manner. These models span heterogeneous technologies and administrative domains. Existing approaches such as YANG, OpenAPI, JSON Schema , ...primarily define the syntax and validation of data, but provide limited mechanisms for collaboration, coordination, and conflict-aware operations across multiple actors.

The Graph-based Meta Schema introduces a unified abstraction that models resources and their relationships as a knowledge graph, connecting data, intent, and operational context. This knowledge graph enables multi-actor automation, where humans, controllers, and services operate on a shared schema using common semantics and consistent operations.

This document defines three complementary elements:

- * A Graph Schema, which specifies how to connect resources modeled using existing data modeling languages (YANG, OpenAPI, JSON Schema, etc.);
- * A set of API operations that provide server-side support for multi-actor collaboration (e.g., conflict management, coordinated deletion, version control, dry-run, and eventing); and

- * A standardized Meta Header carried by each resource, which captures structural and operational metadata—such as identifiers, relationships, ownership, lifecycle hooks (finalizers), labels, annotations, and versioning—used by those operations.

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. Goals and Scope

The goal of the Graph-based Meta Schema is to define a generalized mechanism for describing and managing resources through a graph abstraction, and to standardize the operational semantics required for multi-actor collaboration.

Design principles:

- * Separation of concerns — existing data models (YANG, OpenAPI, JSON Schema) define what a resource is; relationships define how resources interact or depend on one another.
- * Reuse and extensibility — models are reused across domains and extended via typed relations and composable schemas.
- * Simplified data management — graph semantics enable generalized queries, lookups, and mutations (e.g., GraphQL).
- * Server-side multi-actor operations — conflict management, coordinated deletion, versioning, dry-run, and eventing are handled consistently by the API server.
- * Unified metadata via Meta Header — identity, relations, ownership, and lifecycle state are carried within a single metadata structure that accompanies every resource.

3. Conventions and Definitions

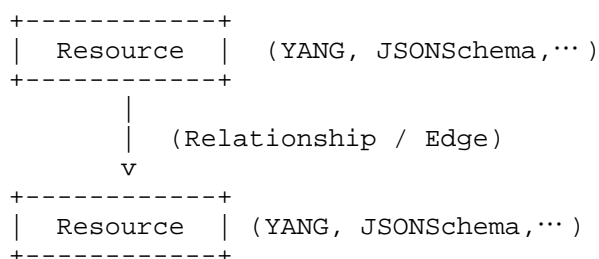
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

4. Conceptual Model

The Meta Schema distinguishes between resources (nodes) and relationships (edges):

- * Resource (Graph Node):
Describes what the entity is. A resource is defined by a schema (e.g., YANG, OpenAPI, JSON Schema) and identified by its Group/Version/Kind/Scope tuple.
- * Relationship (Edge):
Describes how resources interact or depend on each other. Edges connect resources using their identifiers and include metadata and constraints to provide contextual meaning.

This separation of what (resource) and how (relationship) enables the construction of a knowledge graph that links system intent, configuration, and operational context.



5. 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.

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.

6. 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.

7. API Operations

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

Each operation relies on metadata carried in the Meta Header. For example, server-side apply uses managedFields to detect and resolve conflicts; finalizers govern coordinated deletion; version identifiers (generation, resourceVersion) support optimistic concurrency; and relationships/ownership guide cascading behavior.

The standardized operations include:

- * Conflict Management — server-side apply and merge policies.
- * Coordinated Deletion — finalizers and dependent cleanup.
- * Ownership — modeled via relationships
- * Multi-Tenancy — resource scoping by tenant or workspace.
- * Bulk and Dry-Run — batched execution and preview.
- * Streaming and Eventing — consistent event delivery and change propagation.
- * GraphQL Integration — unified query and mutation across resource graphs.

8. Meta Header

Each resource includes a Meta Header, a standardized metadata structure that enables coordination between multiple actors and automation systems. The Meta Header provides a consistent mechanism for storing structural, operational, and user metadata alongside the resource specification.

The Meta Header MUST include the following conceptual components:

- * TypeMeta:
 - Identifies the resource by group, version, and kind.
- * ObjectMeta:
 - Identifiers and lifecycle attributes including name, tenant (if applicable), uid (immutable unique identifier), creationTimestamp, and optional deletionTimestamp.
 - Opaque data: labels and annotations for opaque metadata.
 - Relationships: Directed edges to other resources (e.g., ownership, containment, dependency) expressed with typed targets (group, version, kind, name, and optionally tenant). Ownerships are modelled as relationships
 - Finalizers: Named hooks that delay deletion until cleanup/coordination completes.
 - ManagedFields: Field-level ownership and last-writer metadata for concurrent edits and server-side apply.
 - Versioning: generation and resourceVersion for optimistic concurrency and change tracking.
 - Tenancy: Scope indicators (e.g., global/workspace/tenant) where applicable.

9. Example resource instance including meta header

```
apiVersion: assets.kuid.dev/v1
kind: Device
metadata:
  uid: "c9elf3d1-7f27-4a1b-9f0f-2c9a1b6f7a42"
  name: router1
  tenant: default
  creationTimestamp: "2025-10-31T12:34:56Z"
  # deletionTimestamp is absent until deletion is requested
  resourceVersion: "7"
  generation: 3
  labels:
    purpose: core
  annotations:
    c4o.dev/displayName: "Router-1"
  relationships:
    - name: site
      target:
        group: location.kuid.dev
        version: v1
        kind: Site
        name: site-paris
        tenant: default
  finalizers:
    - cleanup.c4o.dev
  managedFields:
    - manager: controller.network
      operation: Apply
      fields: [ "spec.interfaces", "spec.routing" ]
data: <YANG, ...>
```

10. 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: belongsTo
        target:
          group: location.kuid.dev
          version: v1
          kind: Site
          cardinality: one
      - name: locatedIn
        target:
          group: assets.kuid.dev
          version: v1
          kind: Rack
          cardinality: one
```

Device - (belongsTo) - Site Device - (locatedIn) - Rack

11. IANA Considerations

This document has no IANA actions at this time.

12. 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.

13. 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.
- * [RFC7950] The YANG 1.1 Data Modeling Language

Acknowledgements

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

Authors' Addresses

Wim Henderickx Nokia 50 Copernicuslaan 2018, Antwerp Belgium

Email: wim.henderickx@nokia.com