# Open API semantic tagging for interoperability

STATEMENT:

It is a massive loss that API specification lacks "universal dictionary". Why? Because if there was an universal dictionary then all API publishers would be able to "tag" their content so that API's could better "understand" each other resulting in easier cross-mapping. Additionally if they carried "master data" flag per item, it would allow a single entity (for example a SME) to decide which cloud service / which item is considered as master data (while same item in other linked (synced) cloud platforms) would be synced to that master data.

CHATGPT 4o SPECULATION ON STATEMENT:

**One of the most painful unsolved problems in systems integration**:

**Semantic interoperability** — the ability of different systems to understand each other's data **beyond just structure (schema), down to meaning (semantics)**.

---

## ❌ What Makes Syncing 3rd Party Cloud Platforms So Painful?

1. **Lack of Standard Semantics**:
   - APIs may use different field names for the same concept: `user_id`, `uid`, `id`, `customerNumber`, etc.
   - Even worse: same name, different meaning (`status` can mean login status, project phase, or payment state).
2. **No Master Data Ownership Concept**:
   - APIs rarely declare which system owns the "truth" (master data).
   - Changes may overwrite each other blindly during sync, causing conflict or data loss.
3. **Hardcoded Mappings**:
   - Mapping fields between APIs is done manually, with brittle transformations (Zapier/Make/n8n).
   - If APIs change their contract, everything breaks.
4. **Missing Metadata**:
   - APIs usually don't expose metadata like:
     - "This field is optional."
     - "This field is calculated, don't write to it."
     - "This is the primary key."
     - "This is the master version of this data."

---

## ✅ Your Vision: "Universal Dictionary" + Master Data Flags

You're proposing two key additions:

1. **A Common Vocabulary Layer**:
   - APIs could "tag" their data using shared vocabulary (e.g., `schema.org`, `UN/CEFACT`, `FIBO`, or custom ontologies).
   - E.g., instead of just `user_id`, also tag it with `"schema:identifier.human"`.
2. **Master Data Declaration per Field or Entity**:
   - Every field or item could carry metadata: `isMaster: true/false`
   - Lets you declare which system is the **source of truth** — key for **bidirectional sync**.

This idea parallels **data fabric / knowledge graph / semantic web** principles — but applied **practically** at the **API level**.

---

## 🔗 Are There Efforts in This Direction?

Yes, though fragmented and not yet mainstream:

- **Schema.org / JSON-LD**: Embeds meaning in data (used in SEO, limited API adoption)
- **OpenAPI + x-extensions**: You can manually annotate fields, but no standard vocabulary
- **FHIR (Healthcare)** and **GS1 (Supply Chain)**: Try to standardize semantics for their domains
- **OASIS/UBL**: Business documents with consistent terms
- **Microsoft's Common Data Model (CDM)**: Attempt to normalize semantics across services like Power Platform, Dynamics, Dataverse
- **MDSL (Microservice Data Specification Language)**: Experimental language to describe service interfaces with domain semantics
- **Data Catalogs / Data Mesh Tools**: Offer semantic tagging, but typically internal, not API-facing

---

## 🔄 Is Anything Making Syncing Easier Today?

A few tools and strategies:

- **Integration Platforms (iPaaS)**: Zapier, Make, Workato, Tray.io, n8n — but mapping still manual.
- **Metadata Harvesters**: Like Postman, Stoplight, or SmartAPI try to extract some structure
- **OpenAPI + JSON Schema**: Helps validate structure, not meaning

- **CDPs (Customer Data Platforms)** and **iSaaS**: Some offer identity resolution or master-data rules, but still siloed

---

### 🧠 Summary

Your insight is spot-on. The lack of a **semantic contract** between APIs is a root cause of the fragile and expensive state of integration.

✓ Your proposal:

- Could simplify cross-platform sync
- Would allow smarter, less brittle automations
- Could enable decentralized **data ownership and governance**

🚧 But adoption is a challenge — it requires:

- Standardizing vocabularies (hard in multi-domain world)
- Agreement across vendors
- Possibly new tooling and protocols

QUESTION:

How should we extend the current Open API Specification https://spec.openapis.org/oas/latest.html to enable and support the desired semantic interoperability?

CHATGPT 4o RESPONSE TO THE QUESTION:

To enable **semantic interoperability** within the current OpenAPI Specification (OAS), you would extend the spec by **adding semantic metadata** — ideally using the existing mechanism of **custom vendor extensions (`x-...`)** — in a structured and consistent way.

Here's a concrete breakdown:

---

# ✅ How to Extend OpenAPI for Semantic Interoperability

### 1. Use `x-semantic-tags` or similar extension

You can add semantic tags at the **property level** in OpenAPI schemas to declare **universal meaning** using standard vocabularies (e.g. schema.org, FIBO, CDM, or your own ontology).

### 🔧 Example:

```
yaml
CopyEdit
components:
  schemas:
    User:
      type: object
      properties:
        id:
          type: string
          description: Internal user ID
          x-semantic-tag: "schema:identifier.human"
        email:
          type: string
          format: email
          x-semantic-tag: "schema:email.human"
        is_active:
          type: boolean
          x-semantic-tag: "custom:status.active"
```

---

## 2. Declare Master Data Authority

Introduce a field-level or entity-level declaration like `x-master-data`: This is of course something the owner of the "to be synced API platforms" should be able to decide. Owner should point out which items have "true" for master data. Like entreprenuer in SME paying for 12 different cloud services, willing to use them in sync data wise.

```
yaml
CopyEdit
components:
  schemas:
    User:
      type: object
      x-master-data-owner: true  # This API/service is source of truth
      properties:
        id:
          type: string
          x-master-data: true     # Master field
        email:
          type: string
          x-master-data: false    # Synced, not authoritative
```

You could make this field more expressive, e.g. `"owned" | "synced" | "read-only" | "calculated"`.

---

## 3. Define `x-mapping-candidates` for Crosswalks

If you're designing for integration, you could include potential field mappings to known systems or schemas:

```
yaml
CopyEdit
        customer_id:
          type: string
          x-mapping-candidates:
            - "Salesforce.Contact.Id"
            - "SAP.CustomerNumber"
            - "schema:identifier"
```

This makes it easier for systems to auto-suggest crosswalks or data bridges.

---

### 4. Encourage JSON-LD-style Embedding

You could introduce semantic context in the payload, similar to JSON-LD. OpenAPI 3.1 allows `application/ld+json`as a media type already.

```
yaml
CopyEdit
requestBody:
  content:
    application/ld+json:
      schema:
        $ref: "#/components/schemas/User"
```

---

### 5. Tools / Ecosystem Support

To make this useful:

- Extend tools like Swagger UI, Redoc, and code generators to **visualize semantic tags**.
- Build middleware that **uses semantic tags to auto-match** fields between APIs.
- Add validators that check for conflicts (e.g. two APIs claim to be master of the same tag).

---

## 🌍 Benefits

- Shared meaning across systems → easier integration
- Smarter sync logic (know what's authoritative)
- Easier to automate field mapping and validation
- Lays foundation for decentralized data governance

---

## 🧩 Potential Standards to Build On

- Schema.org
- Common Data Model (CDM)
- FIBO (finance)
- FHIR (healthcare) for advanced metadata strategies
- W3C JSON-LD for embedding context

Here's a **complete OpenAPI 3.1 example document** that uses extensions (`x-semantic-tag`, `x-master-data`, and `x-master-data-owner`) to demonstrate how you could support semantic interoperability and master data governance directly within an API spec.

## 🧩 Scenario: 3 APIs, 3 Different Naming Conventions

### 🎯 Shared Semantic Target: A person/contact record

We'll use this common semantic model:

| Semantic Tag | Meaning |
|---|---|
| `schema:identifier` | Unique ID |
| `schema:name` | Full name |
| `schema:email` | Email address |
| `schema:telephone` | Phone number |
| `schema:jobTitle` | Job title |
| `schema:organization` | Company or employer |

---

## 💼 1. Salesforce API (CRM system)

```
yaml
CopyEdit
SalesforceContact:
  type: object
  properties:
    Id:
      type: string
      x-semantic-tag: schema:identifier
    Name:
      type: string
      x-semantic-tag: schema:name
    Email:
      type: string
      x-semantic-tag: schema:email
    Phone:
      type: string
      x-semantic-tag: schema:telephone
    Title:
```

```
      type: string
      x-semantic-tag: schema:jobTitle
    AccountName:
      type: string
      x-semantic-tag: schema:organization
```

---

## 🛒 2. Shopify API (E-commerce platform)

```
yaml
CopyEdit
ShopifyCustomer:
  type: object
  properties:
    id:
      type: integer
      x-semantic-tag: schema:identifier
    first_name:
      type: string
    last_name:
      type: string
    email:
      type: string
      x-semantic-tag: schema:email
    phone:
      type: string
      x-semantic-tag: schema:telephone
```

To improve semantic mapping:

```
yaml
CopyEdit
    full_name:
      type: string
      x-semantic-tag: schema:name
```

---

## 🏢 3. Microsoft Entra ID (Identity system, formerly Azure AD)

```
yaml
CopyEdit
EntraUser:
  type: object
  properties:
    userPrincipalName:
      type: string
      x-semantic-tag: schema:email
    displayName:
      type: string
      x-semantic-tag: schema:name
    id:
      type: string
      x-semantic-tag: schema:identifier
    jobTitle:
      type: string
      x-semantic-tag: schema:jobTitle
    mobilePhone:
      type: string
      x-semantic-tag: schema:telephone
    companyName:
```

```
      type: string
      x-semantic-tag: schema:organization
```

---

## ✅ Semantic Tagging Results in This Mapping Table:

| Concept | Salesforce | Shopify | Microsoft Entra ID |
|---|---|---|---|
| schema:identifier | `Id` | `id` | `id` |
| schema:name | `Name` | `full_name`* | `displayName` |
| schema:email | `Email` | `email` | `userPrincipalName` |
| schema:telephone | `Phone` | `phone` | `mobilePhone` |
| schema:jobTitle | `Title` | — | `jobTitle` |
| schema:organization | `AccountName` | — | `companyName` |

* `full_name` may need to be synthesized from `first_name` + `last_name`.

---

## 🔍 Why This Matters

- Without semantic tagging, the **same concept is hidden under different names** → hard to map
- With tagging, tools can:
  - **Auto-align fields across APIs**
  - Warn of **missing data** (e.g., no `schema:jobTitle` in Shopify)
  - Enable **low-code sync logic** with confidence in what each field means

## 🧠 Overview: AI-Orchestrated API Sync (Semantic + Master Data)

### ✅ Assumptions

- Each service publishes a **semantically-tagged OpenAPI 3.1 spec**
- Fields are annotated with:
  - `x-semantic-tag`: describing *what* the field means
  - `x-master-data`: whether the field is authoritative
- AI engine has access to:
  - All participating OpenAPI specs
  - Some context/config from the data owner (e.g., sync direction preferences)

---

## 🧩 Step-by-Step Process (High-Level)

### 1. Ingest & Normalize API Specs

AI parses `n` OpenAPI specs with semantic tags:

- Builds a **unified semantic index**, e.g.:

```
json
CopyEdit
{
  "schema:email": {
    "Salesforce": "Email",
    "Shopify": "email",
    "EntraID": "userPrincipalName"
  },
  ...
}
```

- Groups together all representations of the same concept using semantic tag keys.

---

## 2. Detect Master Data Ownership

AI reads `x-master-data: true` flags:

- Example:

```
yaml
CopyEdit
Salesforce:
  Email: x-master-data: true
Shopify:
  email: x-master-data: false
EntraID:
  userPrincipalName: x-master-data: false
```

- Salesforce is identified as **source of truth for emails**, so sync logic should *push from* Salesforce and *pull to*others.

---

## 3. Auto-Generate Mapping Rules

AI generates:

- **Field mappings** across services (via `x-semantic-tag`)
- **Transformation logic** (e.g., join `first_name` + `last_name` → `schema:name`)
- **Directionality rules**:
  - Fields with `x-master-data: true` → *write to others*
  - Fields without `x-master-data` or set to `false` → *read-only or read-preferring*

Example rule:

```
json
CopyEdit
{
  "source": "Salesforce.Email",
```

```
  "targets": [
    "Shopify.email",
    "EntraID.userPrincipalName"
  ],
  "direction": "one-way",
  "onConflict": "overwrite",
  "syncInterval": "daily"
}
```

---

## 4. Build Sync Workflows (e.g. in n8n)

AI uses predefined nodes for HTTP requests, authentication, transformations, and error handling.

Each service:

- GETs records
- Transforms them based on the unified semantic model
- POSTs/PUTs to targets that are *not master*

For example:

- **Salesforce → Shopify + EntraID**
  - Email and Name pushed
- **Shopify → Salesforce**
  - Only `phone` or purchase metadata might be pulled (if it's not mastered by Salesforce)

These workflows are visualized, exported as n8n JSON configs, or deployed via the n8n API.

---

## 5. Monitor and Resolve Conflicts

AI watches:

- Divergences from sync logic
- Repeated overwrites / ping-pong loops (common in naive bi-directional sync)

Uses tags + master flags to:

- Enforce *no overwrite of master fields*
- Suggest better sync rules (e.g., move `phone` master from Shopify to EntraID)

---

## 🔐 Example: Managing `email` and `jobTitle`

- **Salesforce** is `x-master-data` for `email` and `jobTitle`
- **AI** sets up push-only logic for these fields

- **EntraID** has `userPrincipalName`, so AI maps `schema:email → userPrincipalName`
- If EntraID tries to update `email`, AI **blocks or logs** the attempt unless explicitly overridden

---

## ⚙️ Benefits of This AI-Orchestrated Sync

| Feature | Enabled by |
|---|---|
| Auto-matching of fields | `x-semantic-tag` |
| Authority-based syncing | `x-master-data` |
| Minimal manual setup | AI-driven orchestration |
| Scalability to 100s of APIs | Semantic abstraction |
| Conflict prevention | Master data control logic |

---

## 🌍 Final Thoughts

With this model:

- APIs become **self-describing**
- AI becomes a **data sync orchestrator**, not just a middleware
- Businesses can **declare data governance** per field across systems

This is **realistic to implement** today with current OpenAPI 3.1 + tooling + simple AI agents (no deep learning needed).

tomas.westerholm@tietoa.fi