Ontology Examples: Diet Restrictions

Solving Real-World Problems with Knowledge Graphs

The Problem: Can Alice Eat This?

Scenario: Restaurant recommendation system

Challenge:

- Alice is vegetarian
- Bob has peanut allergy
- Carol is lactose intolerant
- David follows halal diet

Question: Which menu items can each person safely eat?

Traditional approach: Hard-coded if-statements (nightmare to maintain!)

Why This Is Hard Without Ontology

```
// Traditional approach - hard to maintain
boolean canEat(Person person, Food food) {
    if (person.isVegetarian() && food.containsMeat())
        return false;
    if (person.getAllergies().contains("peanut")
        && food.getIngredients().contains("peanut"))
        return false;
    if (person.isLactoseIntolerant()
        && food.containsDairy())
        return false;
    // 100 more if statements...
}
```

Problems: Not scalable, hard to update, can't explain why

The Ontology Solution

Key Idea: Model knowledge explicitly

1. **Define domain concepts** (Person, Food, Ingredient, Restriction)
2. **Express relationships** (contains, incompatibleWith, follows)
3. **Add rules and constraints** (reasoning logic)
4. **Query and validate** (SPARQL for querying, SHACL for validation)

Result: Flexible, explainable, maintainable system

Step 1: Define the Ontology

Core Classes

```
@prefix diet: <http://example.org/diet#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

# Main classes
diet:Person rdf:type owl:Class .
diet:Food rdf:type owl:Class .
diet:Ingredient rdf:type owl:Class .
diet:Restriction rdf:type owl:Class .
diet:DietaryPreference rdf:type owl:Class .
```

Food and Ingredient Hierarchy

```
# Ingredient categories
diet:Ingredient rdf:type owl:Class .

diet:Meat rdfs:subClassOf diet:Ingredient .
diet:Dairy rdfs:subClassOf diet:Ingredient .
diet:Nut rdfs:subClassOf diet:Ingredient .
diet:Seafood rdfs:subClassOf diet:Ingredient .
diet:Vegetable rdfs:subClassOf diet:Ingredient .
diet:Grain rdfs:subClassOf diet:Ingredient .

# Specific ingredients
diet:Beef rdfs:subClassOf diet:Meat .
diet:Chicken rdfs:subClassOf diet:Meat .
diet:Pork rdfs:subClassOf diet:Meat .

diet:Milk rdfs:subClassOf diet:Dairy .
diet:Cheese rdfs:subClassOf diet:Dairy .
diet:Butter rdfs:subClassOf diet:Dairy .
```

Restriction Types

```
# Types of restrictions
diet:Restriction rdf:type owl:Class .

diet:Allergy rdfs:subClassOf diet:Restriction .
diet:Religious rdfs:subClassOf diet:Restriction .
diet:Medical rdfs:subClassOf diet:Restriction .
diet:Ethical rdfs:subClassOf diet:Restriction .

# Specific restrictions
diet:PeanutAllergy rdf:type diet:Allergy .
diet:LactoseIntolerance rdf:type diet:Medical .
diet:Vegetarian rdf:type diet:Ethical .
diet:Vegan rdf:type diet:Ethical .
diet:Halal rdf:type diet:Religious .
diet:Kosher rdf:type diet:Religious .
```

Properties (Relationships)

```
# Person-related properties
diet:hasRestriction rdf:type owl:ObjectProperty ;
    rdfs:domain diet:Person ;
    rdfs:range diet:Restriction .

diet:hasAllergy rdf:type owl:ObjectProperty ;
    rdfs:subPropertyOf diet:hasRestriction ;
    rdfs:domain diet:Person ;
    rdfs:range diet:Ingredient .

# Food-related properties
diet:containsIngredient rdf:type owl:ObjectProperty ;
    rdfs:domain diet:Food ;
    rdfs:range diet:Ingredient .

diet:incompatibleWith rdf:type owl:ObjectProperty ;
    rdfs:domain diet:Restriction ;
    rdfs:range diet:Ingredient .
```

Step 2: Add Data Instances

Define People

```
# Alice - Vegetarian
diet:Alice rdf:type diet:Person ;
    diet:hasName "Alice Johnson" ;
    diet:hasRestriction diet:Vegetarian .

# Bob - Peanut allergy
diet:Bob rdf:type diet:Person ;
    diet:hasName "Bob Smith" ;
    diet:hasAllergy diet:Peanut .

# Carol - Lactose intolerant
diet:Carol rdf:type diet:Person ;
    diet:hasName "Carol Lee" ;
    diet:hasRestriction diet:LactoseIntolerance .

# David - Halal
diet:David rdf:type diet:Person ;
    diet:hasName "David Ahmed" ;
```

Define Menu Items

```
# Caesar Salad
diet:CaesarSalad rdf:type diet:Food ;
    diet:hasName "Caesar Salad" ;
    diet:containsIngredient diet:Lettuce ;
    diet:containsIngredient diet:Cheese ;
    diet:containsIngredient diet:Chicken .

# Peanut Butter Sandwich
diet:PBSandwich rdf:type diet:Food ;
    diet:hasName "Peanut Butter Sandwich" ;
    diet:containsIngredient diet:Bread ;
    diet:containsIngredient diet:Peanut .

# Vegetable Stir Fry
diet:VegStirFry rdf:type diet:Food ;
    diet:hasName "Vegetable Stir Fry" ;
    diet:containsIngredient diet:Broccoli ;
    diet:containsIngredient diet:Carrot ;
    diet:containsIngredient diet:SoySauce .
```

Define Restriction Rules

```
# Vegetarian cannot eat meat
diet:Vegetarian diet:incompatibleWith diet:Meat .

# Lactose intolerance incompatible with dairy
diet:LactoseIntolerance diet:incompatibleWith diet:Dairy .

# Halal restrictions
diet:Halal diet:incompatibleWith diet:Pork .
diet:Halal diet:incompatibleWith diet:Alcohol .

# Vegan restrictions (stricter than vegetarian)
diet:Vegan diet:incompatibleWith diet:Meat .
diet:Vegan diet:incompatibleWith diet:Dairy .
diet:Vegan diet:incompatibleWith diet:Egg .
diet:Vegan diet:incompatibleWith diet:Honey .
```

Step 3: Query with SPARQL

Query 1: What can Alice eat?

```
PREFIX diet: <http://example.org/diet#>

SELECT ?food ?foodName
WHERE {
    # Alice is the person
    diet:Alice diet:hasRestriction ?restriction .

    # Get all foods
    ?food rdf:type diet:Food ;
          diet:hasName ?foodName .

    # Check if food contains incompatible ingredients
    FILTER NOT EXISTS {
        ?food diet:containsIngredient ?ingredient .
        ?restriction diet:incompatibleWith ?ingredientCategory .
        ?ingredient rdfs:subClassOf* ?ingredientCategory .
    }
}
```

Query 1 Results

```
Results for Alice (Vegetarian):

✓ Vegetable Stir Fry
✓ Falafel Wrap
✓ Cheese Pizza

✗ Caesar Salad (contains Chicken)
✗ Beef Burger (contains Beef)
```

Explanation: Query finds foods that don't contain ingredients incompatible with vegetarian diet

Query 2: Why can't someone eat a food?

```
PREFIX diet: <http://example.org/diet#>

SELECT ?personName ?foodName ?restriction ?ingredient
WHERE {
    # Specific person and food
    diet:Alice diet:hasName ?personName ;
               diet:hasRestriction ?restriction .

    diet:BeefBurger diet:hasName ?foodName ;
                    diet:containsIngredient ?ingredient .

    # Find the conflict
    ?restriction diet:incompatibleWith ?category .
    ?ingredient rdfs:subClassOf* ?category .
}
```

This provides explainability!

Step 4: Validation with SHACL

What is SHACL?

SHACL (Shapes Constraint Language):

- Validates data against rules
- Ensures data quality
- Catches errors before they cause problems

Think of it as "unit tests" for your knowledge graph

SHACL Shape 1: Person Must Have Name

```
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix diet: <http://example.org/diet#> .

diet:PersonShape
    a sh:NodeShape ;
    sh:targetClass diet:Person ;
    sh:property [
        sh:path diet:hasName ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ;
        sh:minLength 1 ;
        sh:message "Person must have exactly one non-empty name" ;
    ] .
```

SHACL Shape 2: Food Must Have Ingredients

```
diet:FoodShape
    a sh:NodeShape ;
    sh:targetClass diet:Food ;
    sh:property [
        sh:path diet:containsIngredient ;
        sh:minCount 1 ;
        sh:class diet:Ingredient ;
        sh:message "Food must contain at least one ingredient" ;
    ] ;
    sh:property [
        sh:path diet:hasName ;
        sh:minCount 1 ;
        sh:datatype xsd:string ;
        sh:message "Food must have a name" ;
    ] .
```

Testing Validation: Invalid Data

```
# Invalid person - no name
diet:Frank rdf:type diet:Person ;
    diet:hasRestriction diet:Vegetarian .
# ❌ Violation: Person must have a name

# Invalid food - no ingredients
diet:MysteryDish rdf:type diet:Food ;
    diet:hasName "Mystery Dish" .
# ❌ Violation: Food must have at least one ingredient
```

Validation Report Example

```
SHACL Validation Report
=======================

Status: VIOLATION

1. Shape: diet:PersonShape
   Focus Node: diet:Frank
   Path: diet:hasName
   Message: "Person must have exactly one non-empty name"

2. Shape: diet:FoodShape
   Focus Node: diet:MysteryDish
   Path: diet:containsIngredient
   Message: "Food must contain at least one ingredient"

Total Violations: 2
```

Real Implementation: Python Example

```
from rdflib import Graph, Namespace
from rdflib.plugins.sparql import prepareQuery

# Load ontology
g = Graph()
g.parse("diet_ontology.ttl", format="turtle")

# Define namespace
DIET = Namespace("http://example.org/diet#")

# Query: What can Alice eat?
query = prepareQuery("""
    PREFIX diet: <http://example.org/diet#>
    SELECT ?foodName WHERE {
        diet:Alice diet:hasRestriction ?r .
        ?food rdf:type diet:Food ;
              diet:hasName ?foodName .
        FILTER NOT EXISTS {
            ?food diet:containsIngredient ?i .
            ?r diet:incompatibleWith ?c .
            ?i rdfs:subClassOf* ?c .
        }
    }
""", initNs={"diet": DIET})

results = g.query(query)
for row in results:
    print(f"✓ {row.foodName}")
```

Key Takeaways

✓ Ontology makes domain knowledge explicit
✓ SPARQL provides powerful querying
✓ SHACL ensures data quality
✓ Explainable results (can show why)
✓ Easy to extend (add new restrictions)
✓ Integration-friendly (merge data sources)

Diet restrictions is just one example - same patterns apply to many domains!