Ontology and Validation for Geocaching

Author: Tomáš Vrána

Link: https://github.com/xvrat01/GD-SP

1. Problem Description and Goals

Geocaching is a game based on finding hidden containers ("caches") using GPS coordinates. Although it takes place in the real world, its data is highly structured and governed by strict "Guidelines.

The goal of this project was to create an **ontology** (owl + rdfs) and a **set of validation constraints** (shacl) for the Geocaching domain.

A specific problem addressed by this project is **collision checking**. Every physical cache must be at least 161 meters (0.1 miles) away from any other physical cache or its physical stage. This check is relatively complicated, because it requires comparing every entity with all the other ones. Also, there must be the ability to distinguish which points belong to the same object (where collision is permitted) and which belong to another one.

2. Approach and Proposed Solution

2.1. Tools Used

For the main part of the project, I used Protégé. For the repetitive tasks I used plain text editor. For the writing of the repetitive blocks of "code" (eg. cache attributes) I used LLMs – ChatGPT and Gemini.

2.2. Ontology and Mapping to Schema.org

Instead of creating a proprietary vocabulary "from scratch," I chose to extend the existing **Schema.org** standard.

- The classes gc:GeoCache and gc:Waypoint are subclasses of schema:Place.
- The gc:Geocacher class (player) inherits from schema:Person.
- Then I used schema:Comment for gc:GeoCacheLog and schema:DefinedTerm for classes that contain predefined constants (eg. cache attribute type and cache type)
- The class schema:GeoCoordinates I used to assign coordinates to geocache and waypoints.

- **GeoSPARQL** (geo:asWKT) is used to define coordinates, enabling future advanced geographic querying.

2.3. Classes

In the ontology I made essential entities for geocaching. Every class is mapped to schema.org.

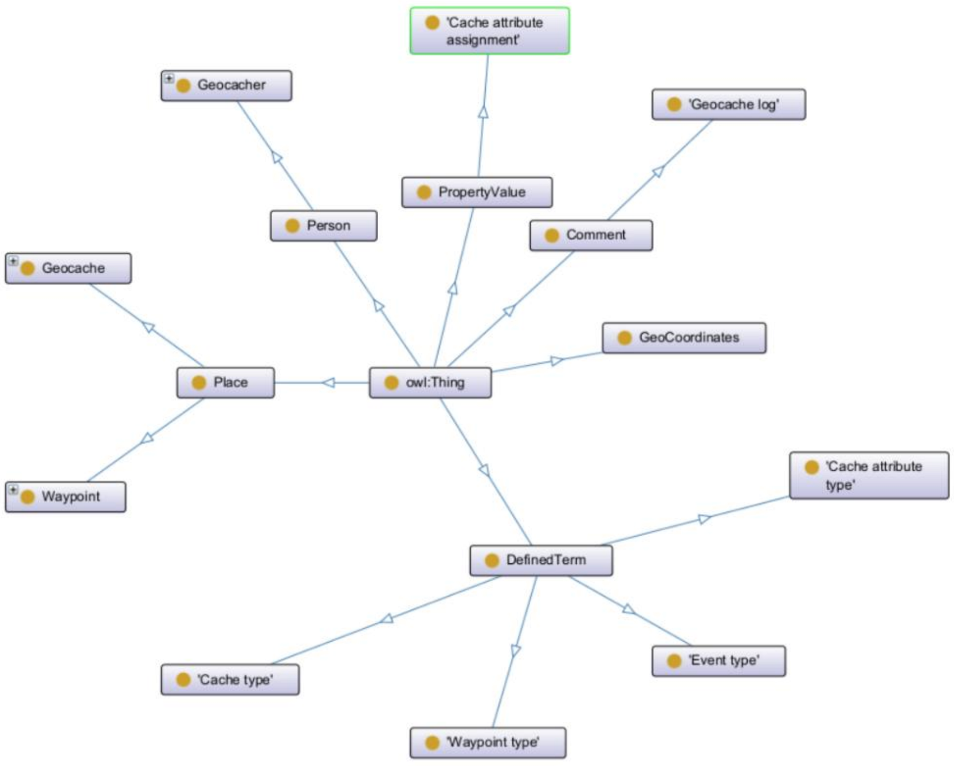


Figure 1 - Graph of classes used in ontology

2.4. Geocache Attribute System

I would like to point out the geocache attribute system. For the cache attributes (e.g., "wheelchair accessible," "night cache") I designed the gc:CacheAttributeType with predefined individuals. Then I designed gc:CacheAttributeAssignment class, that allows any number of attributes to be assigned to a cache alongside with their state (positive/negative). Validation then ensures uniqueness (a cache cannot have the same attribute type assigned twice).

The "cache type", "Event type" and "Waypoint type" works the same as "Cache attribute type". In shacl I used sh:class to ensure that the types provided are valid.

3. Implementation of Validation (SHACL) and Key Challenges

Aside of the ontology, I made **shacl validation** for data. I encountered several technical challenges here that required original solutions.

3.1. Conditional Validation for Events

Geocaching guidelines states that if the type of the cache is “event”, there must be date and other entities. There are few more conditional rules like that. For that purpose, I used conditional validation. I use sh:sparql constrain in shacle to apply these NodeShapes only to the entities that meet the specific rules.

3.2. Problem: The 161-meter Proximity Rule

The complex part of the validation part of the project was implementing the rule prohibiting the placement of a physical cache closer than 161 meters to another.

I wanted to use GeoSPARQL to that. There are specific functions to that purpose. Therefore I used the GeoSPARQL type of coordinates. Standard SHACL validators in editors (such as Protege) often lack support for GeoSPARQL functions, so I needed to use mathematical approach to calculate it. Firstly, I tried the haversine formula, but it used goniometric functions (from math ontology). I found out, that the protégé lack support of that ontology too. At the end I used simplified math formula, that use approximation.

4. Sample SPARQL queries

The queries were executed on the mock data (geo-data.ttl) I made for this purpose.

4.1. Query 1

Query in natural language: “*Find all geocaches, and outpu their names, their type, their owner and their additional owner (the last one is optional).*”

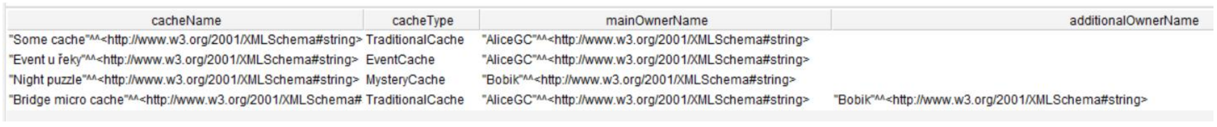
Query in SPARQL:

```
PREFIX gc: <http://example.org/geocaching#>
PREFIX schema: <https://schema.org/>

SELECT ?cacheName ?cacheType ?mainOwnerName ?additionalOwnerName
WHERE {
  ?cache a gc:GeoCache ;
         schema:name ?cacheName ;
         gc:mainOwner ?owner ;
         gc:cacheType ?cacheType.
  ?owner schema:name ?mainOwnerName .

  OPTIONAL {
    ?cache gc:additionalOwner ?additionalOwner .
    ?additionalOwner schema:name ?additionalOwnerName .
  }
}
```

Output:



cacheName	cacheType	mainOwnerName	additionalOwnerName
"Some cache"^^<http://www.w3.org/2001/XMLSchema#string>	TraditionalCache	"AliceGC"^^<http://www.w3.org/2001/XMLSchema#string>	
"Event u řeky"^^<http://www.w3.org/2001/XMLSchema#string>	EventCache	"AliceGC"^^<http://www.w3.org/2001/XMLSchema#string>	
"Night puzzle"^^<http://www.w3.org/2001/XMLSchema#string>	MysteryCache	"Bobik"^^<http://www.w3.org/2001/XMLSchema#string>	
"Bridge micro cache"^^<http://www.w3.org/2001/XMLSchema#	TraditionalCache	"AliceGC"^^<http://www.w3.org/2001/XMLSchema#string>	"Bobik"^^<http://www.w3.org/2001/XMLSchema#string>

Figure 2 - Query 1 - Output

4.2. Query 2

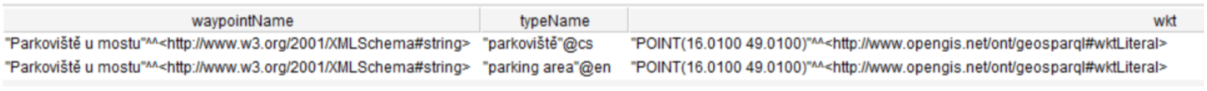
Query in natural language: "*List all Waypoints for a specific Cache and output their name, typeName and coordinates*."

Query in SPARQL:

```
PREFIX gc: <http://example.org/geocaching#>
PREFIX schema: <https://schema.org/>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>

SELECT ?waypointName ?typeName ?wkt
WHERE {
  ?cache schema:name "Bridge micro cache"^^xsd:string .
  ?cache gc:hasWaypoint ?wp.
  ?wp schema:name ?waypointName;
    gc:waypointType ?type;
    schema:geo ?geoNode .
  ?type schema:name ?typeName .
  ?geoNode geo:asWKT ?wkt .
}
```

Output:



waypointName	typeName	wkt
"Parkoviště u mostu"^^<http://www.w3.org/2001/XMLSchema#string>	"parkoviště"@cs	"POINT(16.0100 49.0100)"^^<http://www.opengis.net/ont/geosparql#wktLiteral>
"Parkoviště u mostu"^^<http://www.w3.org/2001/XMLSchema#string>	"parking area"@en	"POINT(16.0100 49.0100)"^^<http://www.opengis.net/ont/geosparql#wktLiteral>

Figure 3 - Query 2 - Output

4.3. Query 3

Query in natural language: "*Find all geocaches with specific attribute (Wheelchair accessible = True) and output their name and difficulty.*"

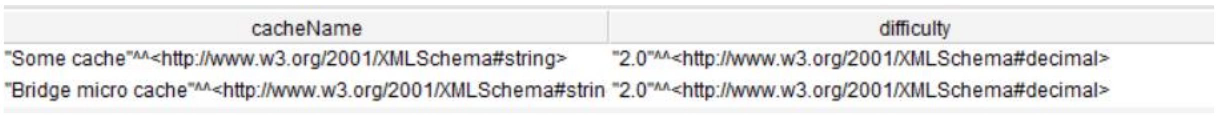
Query in SPARQL:

```
PREFIX gc: <http://example.org/geocaching#>
PREFIX schema: <https://schema.org/>

SELECT ?cacheName ?difficulty
WHERE {
  ?cache a gc:GeoCache ;
         schema:name ?cacheName ;
         gc:difficulty ?difficulty ;
         gc:hasAttribute ?assignment .
  ?assignment gc:attributeType gc:Attr_WheelchairAccessible ;
```

```
               gc:attributeState true .
}
```

Output:



cacheName	difficulty
"Some cache"^^<http://www.w3.org/2001/XMLSchema#string>	"2.0"^^<http://www.w3.org/2001/XMLSchema#decimal>
"Bridge micro cache"^^<http://www.w3.org/2001/XMLSchema#strin	"2.0"^^<http://www.w3.org/2001/XMLSchema#decimal>

5. Conclusion

The project demonstrates that graph technologies are suitable for modeling Geocaching rules. The first part of the project consist of the ontology (geo-ontology.ttl). The second part is the creation of a SHACL rules (geo-shapes.ttl). For the test purposes I made some data (geo-data.ttl) to test the functionality.

The issue of missing GeoSPARQL function support in common editors was solved by implementing a mathematical functions within a basic SPARQL. This enables to perform distance-checks to be performed without any other ontologies used, but at the expense of precision.