Semantic Web Project Group 59

Lily Maechling, Christoph Moser, Katherine Lasonde, Matti Teodorascu

October 2021

Integrated First and Second Milestone

Goal

Our application build is primarily focused on the planets in our Solar System. There are eight planets in the Solar system – Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, and Neptune – which have been abundantly studied in isolation and groups. The wealth of data about the planets is sprinkled across the world wide web. The proposed application will extract, clean, combine, and present the many data sets intuitively to the user.

In particular, the primary goal of this application is to allow users to identify and analyze relationships and trends between Solar System planets. This is done by identifying various data sets on the web relating planets to properties such as 'distance from the Sun,' 'orbital velocity,' 'mass,' 'diameter,' and more. One example of a trend that a user may identify based on our data is that a planet's size and mass will determine its gravitational pull. There is also a focus on data related to the moons of the planets.

Stakeholders

The application is primarily intended for NASA scientists or other professional astronomers researching planets within the Solar System. However, anyone desiring to learn more about the planets may be interested in utilizing our application. The Sun and the Solar system planets formed together, approximately 4.6 billion years ago, from a collapse of a solar nebula and the four forces (strong nuclear, weak nuclear, gravity, electromagnetic). Thus, their properties are inherently integrated and related. To best understand the Solar system, users must study planets within the context of each other as they are constantly affecting each other. The application will help these users retrieve specific properties about planets in an accessible manner and make conclusions about those properties.

Design

Visualizing the spacial properties of celestial bodies, solar systems, galaxies and the likes is not straightforward because of the extreme figures in units that humans are used to, often making it difficult to fully grasp the sizes of and distances between such objects. One way to combat this issue is to use special units such as the astronomical unit (AU) which corresponds to the distance between the sun and the earth, or light years (LY).

Other options include plotting figures on graphs with exponential axes or overall using exponential numbers in the analysis. For objects that circumnavigate a common center on more or less the same vertical axis (such as the planets of our solar system circumnavigate the sun), also a "top-down" image of the objects can give a good overview of the distances inbetween the objects.

Ontologies

`https://dbpedia.org/ontology/Planet`
`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
`http://www.w3.org/2000/01/rdf-schema#`
`http://xmlns.com/foaf/spec/`

We plan on using the above ontologies. The first ontology comes from /urldbpedia.org and outlines various relationships and attributes of a planet. Some important datatype properties provided by this ontology include dbo:density, dbo:mass, dbo:surfaceArea, and dbo:temperature. All of these properties are important to correctly model a given planet with its various data fields. Next, we plan on using two generic ontologies of rdfs and rdf. These ontologies will provide us important properties to restrict and further describe our data. Some properties we plan on using include rdf:type, rdfs:label, rdfs:subClassOf, and rdfs:subPropertyOf. Finally, we plan on using the foaf ontology to represent relational properties between planets and their satellites.

Data

General Planet Data (tabular): `https://nssdc.gsfc.nasa.gov/planetary/factsheet/`
General Natural Satellite (i.e. Moons) Data (tabular): `https://en.wikipedia.org/wiki/List_of_natural_satellites`
SPARQL Endpoint: `http://dbpedia.org/sparql`

We plan on using the above data sources to create the instances for our analysis. The first two data sources are tabular and outline important data describing planets and their satellites. The NASA data set contains all numerical

data describing the 8 planets in our solar system. This data includes mass, density, gravity, escape velocity, and orbital velocity. The second data set contains data on satellites for each planet in our solar system. It includes parent, name, radius, and discovery year. Finally, we will query the dbpedia sparql endpoint for missing planet data and to form some of the object properties.

Domain & Scope

Our team selected the domain of the Solar System, with a particular emphasis on the eight planets and satellites within the system. We chose this domain as there is a plethora of data online about the Solar System, but there is no easily accessible Solar system ontology. Therefore, the ontology will include relevant information on planet properties, satellites properties, and all combinations of relationships between planets and satellites.

The scope of the application is to retrieve data and relations within celestial bodies in the solar system for people who want to learn more about outer space. Below are example questions that our application could answer:

- Which planet has the largest volume?
- What is the range of planet densities in the solar system?
- Which planet has the largest volume?
- Which planets are made of mainly gas?
- Which three satellites are closest to the Earth (in AU)?
- What are the hottest and coldest temperatures in the solar system, and which planets have these records?
- Does Saturn or Jupiter have a faster orbital velocity?
- Which satellite has the fastest orbital velocity?
- What satellites are related to Neptune?

We chose this scope to give the user easily accessible and clear information about relationships in the Solar System.

Methodology

Our main tool to make the data set into an ontology was GraphDB. We are using two different sets of tables. One of the planets of the solar system and of the moons of those planets. Both of those data sets were converted into RDF models using the built-in feature of GraphDB, OntoRefine. After importing the tables in the application, we cleaned the data sets like removing unwanted rows and columns. After the tables were all set, the first classes were made, Planet and Moon. We set each column that contained a planet in the database as an

individual for the class Planet, and did the same for the moons. There is a lot of data of the celestial bodies in the records, which needs to be processed. First we made the datatype properties for both classes. Those are constants like mass or radius. At last, the relationship between the planet and the moons, which is the object property of hasSatellite. After inserting both tables into the graph, the database did not work. Yet after importing one of the tables as a RDF, which we created by using OntoRefine, the ontology functioned properly.

Conceptualization & Drawing

The two main classes in the ontology are planet and moon. Planets are then divided even further into a wide range of subclasses like DwarfPlanets, SuperiorPlanet (planets that are bigger than earth), IceGiant, etc.
In the following list, the object properties are listed with their restrictions and the concepts for them are explained

- hasNeighbor ... a symmetric property that connects two neighboring planets
- hasSatellite ... a restrictionless property that connects a planet to a moon. Inverse of orbitsPlanet
- largerThan ... a transitive property that is used to signify that a planet is larger than another. It is the inverse of smallerThan
- orbitsPlanet ... a restrictionless property that connects a moon to its planet. Inverse of hasSatellite
- smallerThan ... a transitive Property that is used to signify that a planet is smaller than another. Invers of biggerThan.

The following list describes the used dataProperties

- hasDiscoveryYear ... the discovery year of an object. String
- hasMass ... the mass of an object. String
- hasMeanRadius ... the mean radius of an object. String
- hasMoons ... the number of moons a certain planet has. String
- hasOrbitalPeriod ... the orbitalPeriod of an object. String
- hasOrbitalSemiMajorAxis ... the OrbitalSemiMajorAxis of an object. String
- hasRotationPeriod ... the rotation period of an object. String
- hasSiderealPeriod ... the sidereal period of an object. String

The following figure shows a visualization of the ontology. Not depicted for some more clarity are the moons.

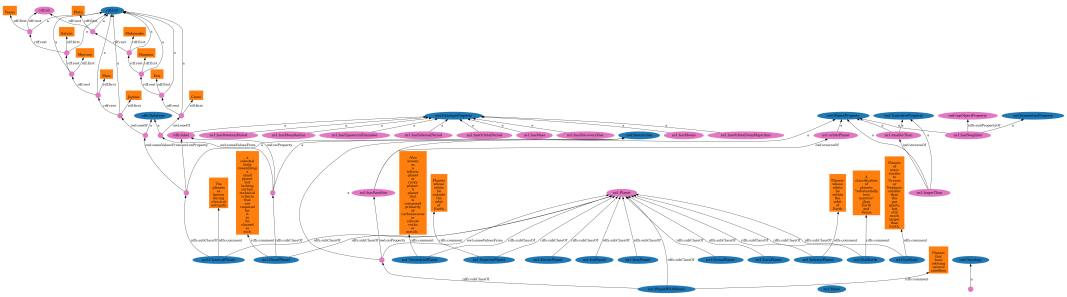

Figure 1: Ontology Visualization

OWL Ontology

Our ontology describes our solar system using two main classes: planets and satellites. Within these two classes, our ontology contains multiple sub-classes and instances of both. Furthermore, each of these instances are described by datatype properties, such as, pl:hasDiscoveryYear, pl:hasMeanRadius, pl:hasMass, ect, and object properties, such as, pl:hasMoon. The planet sub-classes are split by different planet categories, such as dwarf, ice giant, terrestrial planet, and classical planets. Attached is the documentation for the different planet classifications (`https://en.wikipedia.org/wiki/List_of_planet_types`). We have then used the ontology to enforce restrictions. We implemented the following class restrictions:

- For the PlanetWithMoons class, ex:hasSatellite some Planet
- For the DwarfPlanet class,
  rdfs:label some {"Ceres"@en, "Pluto"@en, "Makemake"@en, "Haumea"@en, "Eris"@en}
- For the ClassicalPlanet class,
  rdfs:label some {"Mars"@en, "Jupiter"@en, "Venus"@en, "Saturn"@en, "Mercury"@en}

Integration

In our ontology we have two main classes, Planet and Moon. In our two tabular data sets, there are records about every planet and every moon which is a satellite of said planet. The data are mainly constants of the celestial bodies, constants in the form of mainly integers. So to be able to construct the data in the ontology we made datatype properties which we attached to the individuals in the classes. The constants of the planets in the dataset are now also in the relation of the planets in the ontology. Furthermore, our main goal of the ontology is to visualize the solar system and its properties in a proper way. Therefore, it is also useful to connect the moons to the planets which are

their satellites. We created the hasSatellite object property for planets to be able to see which moons belong to which planets.

(Question to TA: could you elaborate the description of the integration section?)

Inferences

By providing our ontology with certain class restrictions, the reasoner is able to make meaningful inferences among the data set.

One example of a meaningful inference that the reasoner makes is exemplified by Figure 2 in the appendix. Data within our data set can have the **hasSatellite** or **orbitsPlanet**. These two properties are inversely related, meaning that if planet A has a satellite planet B, then planet B must orbit planet A. In our data set, the planet Weywot orbits the planet Quaoar. Thus, the reasoner is able to inference that Quaoar has the satellite planet Weywot.

Another property within our data set is **largerThan**. The **largerThan** (as well as **smallerThan**) property is transitive. Thus, if planet A is larger than planet B, and planet B is larger than planet C, then planet A must be larger than planet C. In figure 3 in the appendix, the reasoner is able to infer than Jupiter isLarger than Earth. This is because the data set details that Jupiter is largerThan Saturn, and the object Saturn is largerThan Earth.

Similarly, any planet which has a satellite planet must be **largerThan** their satellite planet due to laws of gravity. (And any planet which orbits another planet is **smallerThan** the planet which it orbits.) Thus, as demonstrated by Figure 4 in the appendix, since the planet Eris has the satellite planet Dysnomia, then Eris must be **largerThan** Dysnomia.

Finally, objects in the data set may have the reflexive property **isNeighbor**. In other words, if planet A is a neighbor of planet B, then planet B must be a neighbor of planet A. As demonstrated by Figure 5, the ontology is able to infer that Mars is a neighbor of Earth, since it knows that Earth is a neighbor of Mars.

SPARQL Queries

The following SPARQL queries give some examples of information that can be retrieved from the ontology.

1. Query 1 outputs the discovery year and optionally the EquatorialDiameter of all planets.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex: <http://example/pl/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
select *
where
{
```

```
   ?moon rdf:type ex:Moon .
   ?moon ex:hasDiscoveryYear ?year .
    OPTIONAL {?moon ex:hasEquatorialDiameter ?diameter }  .
}
```

Listing 1: Query 1

2. Query 2 outputs the average number, the min and the max number of moons that a single planet has.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex: <http://example/pl/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

#select the average, min and max values
SELECT (AVG(?no) as ?average) (MIN(?no) as ?min) (MAX(?no) as ?max)
WHERE
{
#select number of moons per planet in subquery
SELECT ?planet (COUNT(?planet) as ?no)
WHERE
{
    ?planet ex:hasSatellite ?moon .
}GROUP BY ?planet
}
```

Listing 2: Query 1

Data Visualization

Appendix

Figure 2: Inference example 1 - hasSatellite

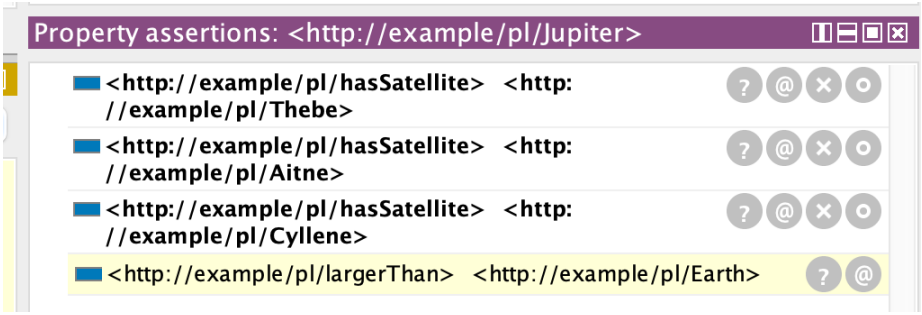

Figure 3: Inference example 2 - largerThan (1)

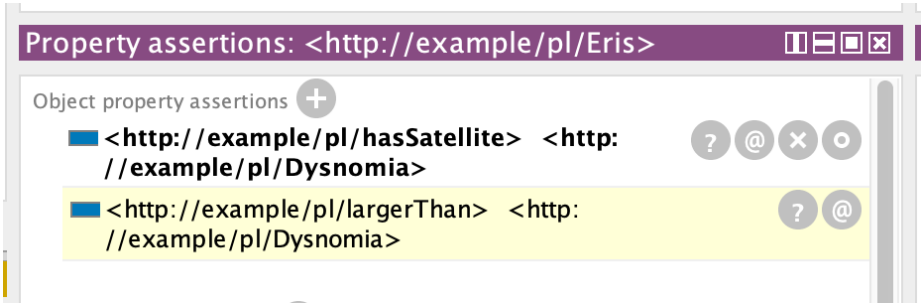

Figure 4: Inference example 3 - largerThan (2)

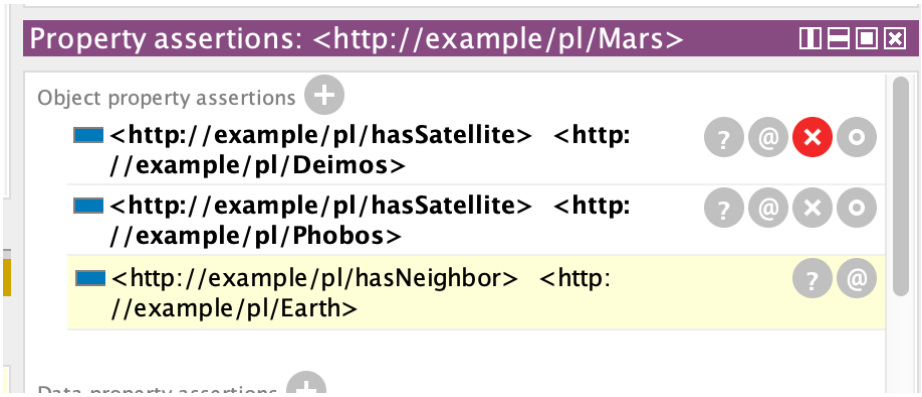

Figure 5: Inference example 4 - hasNeighbor