Weather Phenomenon Prediction Using Semantic Web

Submitted by:
Raktim Kumar Roy
ID: 2015-02-96-005
Spring 2017

Supervised by:
Dr. Mohammad Rezwanul Huq
Assistant Professor
Department of Computer Science and Engineering
East West University

The project has been submitted to the Department of Computer Science and Engineering at East West University in the partial fulfillment of the requirement for the degree of Masters in Computer Science and Engineering.

East West University
Dept. of Computer Science and Engineering
Spring 2017

Declaration

The project has been submitted to the department of computer science and engineering at East West University in the partial fulfillment of the requirement for the degree of Masters in Computer Science and Engineering performed by me under supervision of Dr. Mohammad Rezwanul Huq, Assistant Professor, Dept. of CSE at East West University. This is also need to certify that, the project work under the course ‘Master’s Project (CSE 597)’

I, hereby declare that this project has not submitted elsewhere for the requirement of any degree or diploma or any other purpose.

Signature of the Candidate

Raktim Kumar Roy

Letter of Acceptance

This project is entitled "Weather Phenomenon prediction Using Semantic Web Technology" submitted by Raktim Kumar Roy ID: 2015-02-96-005 to the Department of CSE, East West University, Dhaka-1212, is accepted by the Department for the partial fulfillment of requirements for the degree of MS in CSE April, 2017

Board of Examiners:

Supervisor:

Dr. Mohammad Rezwanul Huq
Assistant Professor
Department of Computer Science & Engineering
East West University
Dhaka, Bangladesh

Chairperson:

Dr. Ahmed Wasif Reza
Associate Professor and Chairperson (acting)
Department of Computer Science & Engineering
East West University
Dhaka, Bangladesh

Acknowledgement

I would like to pay our gratitude to the Almighty God who creates me with not only the ability to build up this Project but also give me power of patient.

I am obliged and thankful to my project supervisor Dr. Mohammad Rezwanul Huq, Assistant Professor, Department of Computer Science and Engineering, East West University for his continuous encouragement, motivation and professional guideline during the work of the project which has proven to be an integral part of it. Without his valuable support and guild line, this project could not elevate up this level of development from my point of view.

I would like to thank all the faculty members, Dept. of CSE, East West University for their valuable time spend in requirement analysis and evaluation of the project work.

I would like to express our sincere cordial gratitude to the people those who have supported me directly, purveyed mental encouragement, evaluated and criticized my work in several phases during the development of this project and for preparing the dissertation indirectly.

Raktim Kumar Roy

Abstract

Weather phenomenon analysis is the application of current technology and science to predict the accurate weather condition for planning our day to day activities. Weather attribute such as temperature, precipitation, wind speed are affected in agriculture, air traffic, marine, forestry, severe weather alerts and advisories, military applications and utility companies. However the process of weather phenomenon is one of the complex areas in meteorology. As weather information is collected from various systems with different formats and parameters. Such type of data is lying in different sources and in heterogeneous format, which are challenging to be integrated in one platform into the knowledge domain. Hence the data source need to be aligned in order to facilitate the smooth integration and to achieve this would involve various processing. There is a need to accomplish unified integration of heterogeneous environments (data sources) and to provide worldwide access to the system. The heterogeneity issues needs to be minimized to arrive a common understanding and decision making by various agencies, research institutions and application areas.

In this paper I focus on a novel framework which is proposed for integrating heterogeneous data sources in a single platform, using semantic web techniques. Here I build a ontology which has been developed for aligning, consisting of all possible concepts, attributes and relations for weather phenomenon domain to provide knowledge using semantic relations.

Table of Content

Chapter 1
Introduction

1.1 Background Study

Bangladesh is one of the largest deltas in the world which is highly vulnerable to natural disasters because of it flat low-lying plain land made up of alluvial soil having small hilly area in the northeast and southeast regions. The great Himalayan Range is to the north and the vast Bay of Bengal is on the south. Due to geo-location Bangladesh face different types of natural disasters. The most common natural disasters in Bangladesh which includes cyclones and associated storm surge, flood, flash flood, severe thunderstorm, Tornado, heavy rainfall, heat wave, cold wave, dense fog etc. Such type's natural hazards may cause huge life loss, damage of properties or demolish the agriculture product. These methods are having drawbacks about the weather attribute which are related to the natural disasters.

Weather phenomenon is a kind of prominent application for various sectors such as agriculture, aviation, military, power industries, forestry, and general public. Agriculturalists require this information to plan their planting and harvesting the crops in the safest way. Aviation sector would require the local weather information to schedule or operate their carriers. Weather forecasting enables us to take precautionary measures in our day to day life, from incidental climate change.

Various websites are providing weather information. Such as Bangladesh Metrological Department, AccuWeather, Open weather map and so on. The format of data, naming convention, measurement units are different for each of the weather systems [Kalra G et al., 2005]. To establish uniformity in terminology and interoperability of data, heterogeneity of data should be minimized by representing data in a common format like XML or RDF.

To utilize the weather information effectively semantic interoperability between different weather systems needs to be addressed. Information technologies are under extraordinarily rapid progress, and a number of them have not been applied to environmental management yet. Semantic web technology addresses this issue by providing solution through ontologies [Berners-Lee T et al., 2001]. Ontology [Noy NFetal., 2001] which provides a convenient vocabulary to represent meaningful specification of data and ontology mapping is the methodology provided for achieving semantic interoperability which is the capability of different information systems to communicate information consistent with the intended meaning of the encoded information.

In this paper I build a ontology on the weather phenomenon domain. Which show how different weather attribute and their phenomenon are related to different weather condition? Building ontology is a very difficult task. Various people build ontology in different way to give meaning to their respective domain. For proper validation an expert opinion is required to determine the dataset on a particular domain. To build this project I had take the help of the thesis paper that had been build on "**Weather Ontology for Predictive Control in Smart Homes published by Paul Staroch"[1]**. The weather attribute which are associated with the weather conditions are taken from various weather related Web Sites. Here I focus on the semantic web technologies in the developed for the domain of weather phenomenon. The main goal of my project is to build the weather phenomenon using ontology based on Web Ontology Language (OWL) which provides a convenient vocabulary abstracting from specific aspects related to the data sources. And it is connected to the underlying relational databases using R2RML mappings. Here I am using H2 Console database which is a relational database management system written in Java. Then data set is mapped according to the vocabulary of my domain concept using an open-source Ontology Based Data Access system (OBDA). Then the ontology is queried in SPARQL(Protocol and RDF Query Language), the OBDA system exploits the mappings to retrieve elements from the data sources and construct the answers expected by the user define.

1.2 Problem Statement

The aim of this project is the development of a data model for weather data which will be utilized in the detecting weather attributes for the related weather condition. Apart from the detecting weather attributes, this model covers previous weather data to determine the max and min temperature, humidity, wind speed, precipitation for a period of days according to the Bangladesh weather data History. Here I am choosing ontology based techniques rather than relational database because huge amount of weather data is very difficult to manage in relational database. In relational every table shares at least one field with another table in one-to-one, one-to-many or many-to-many relationships. To build up a relation to each table is very complex task and time consuming. In huge data set query also become so complex to get accurate result. On the other hand ontology is so powerful to manage the huge data set. Ontology provides the convenient meaning to each data set. We can also easily access the heterogeneous schema, data formats simultaneously and also have other benefit as well. For that reason I have choose Ontology based technique rather than relational database. The main goal of this project to identify the weather attributes for the certain weather condition. Below I give the list of weather condition that I worked on this project.

1. Cyclone.
2. Thunderstorm
3. Foggy weather
4. Hail precipitation
5. Very humid weather.
6. Drizzle rain.

7. Weather condition winter season.
8. Max, Min Temperature for period of days.

We begin with the Introduction and problem statement of the project that I work on. We then describe the related work about the semantic technologies, ontology, OWL, relational database and open database access in chapter 2. In chapter 3 we describe the weather data source which represent about the data source where we collected the weather data. In chapter 4 we describe the methodology of system architecture, methodology of development ontology and the life cycle of ontology. In chapter 5 describes the whole implementation process of this project. Chapter 6 describes the conclusion and future work.

Chapter 2
Related work

The first part of this chapter covers the foundations the work of this project builds upon. It focuses on basic part that gives introductions into all relevant topics, e.g. ontologies and associated modeling languages such as RDF, RDFS, and OWL. Furthermore, it discusses the basic concept of weather attribute that are connected to the weather condition uses a knowledge base built upon a set of ontologies.

The second part is about the existing work regarding weather data applied to determine the weather conditions. Additionally in this chapter I try to elaborately discus the different part of semantic technology that is related to this project.

2.1 Foundations

This chapter presents the foundations the work in this thesis builds upon: Semantic Web, Ontology, OWL,RDF, RDFS, Reasoning, Ontop Mapping and Relational database.

2.1.1 Semantic Web

Semantic Web extends the web with machine interpretable meaning, thus establishing data integration and sharing, and interoperability amongst interconnected machines[11]. The semantic web concept is based on the Resource Description Framework (RDF) which enables linking and merging of relations between entities from multiple resources in the web via Internationalized Resource Identifiers (IRI). RDF Schema (RDFS) and ontologies provide vocabulary for modeling and describing RDF data. Using semantic technologies we can manipulate the heterogeneous data formats and brings them in single platform. It brings a new era to access the data that are in different schema or formats.

2.1.2 Ontology

In computer science, ontology represents knowledge as a set of concepts in a certain domain and relationships between pairs of concepts [10]. The basic elements of ontologies – concepts, properties, and relations – comprise a shared vocabulary which can be used to model a certain domain. Each object that is mapped into an ontology is represented by an individual (also known as object). Individuals of the same type can be defined to be instances of concepts (also called classes). Both classes and individuals can have attributes that specify their characteristics and properties. Two arbitrary classes or individuals can be related to each other via a relation. Furthermore, ontologies may contain function terms (structures formed from relations that can be used in place of terms in statements), restrictions (descriptions of what must be true for additional knowledge to be accepted), rules (statements in if-then notation that describe logical inferences that can be drawn), axioms (core knowledge of the ontology that is known to be true), and events (changes to attributes or relations).

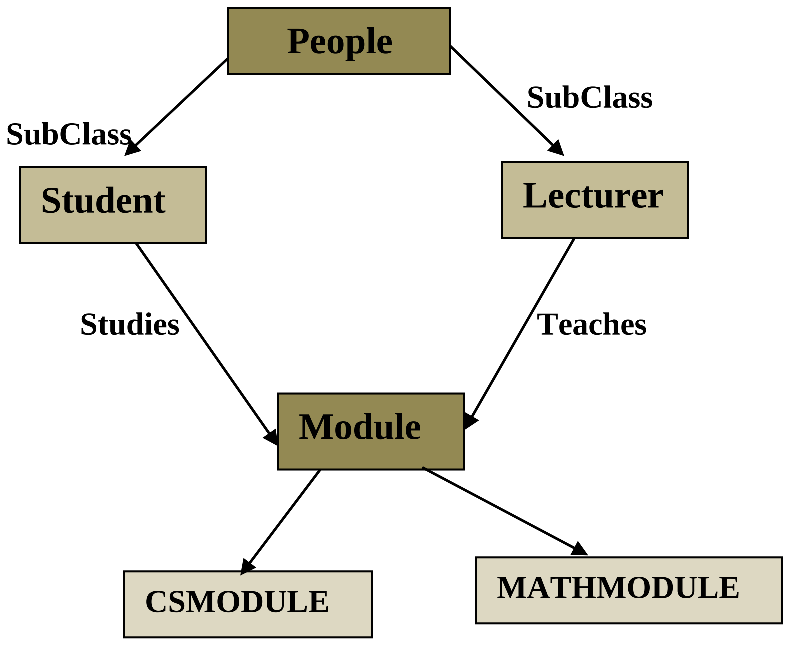


Figure 2.1: Example of a simple ontological model based on University

Figure 2.1 illustrates the aforementioned elements in a simple ontology based on University. People, Student, Lecturer and Module are concepts. People are a superClass and Student and Lecturer is a Subclass of People. SubClass of is a property that defines one concept to be a sub-concept of another concept (i.e. B is subclass of A states that every instance of B is also an instance of A). Student is connected to the Module class by the object properties Studies and the Lecturer is connected to the Module class by the object properties Teaches. Module Class has two Individuals CSModule and MathModule.

This model states the following facts: Student Studies CSModule and MathModule and Lecturer Teaches the CSModule and MathModule.

An important feature of an ontology is the support of automatic reasoning to deduce facts that are not explicitly stated in the data model from the given information. In order to make Reasoning possible, the semantics of data models in ontologies (including OWL) are often based on Description Logics . These are a family of logics consisting of decidable parts of first-order predicate logic. After the Reasoning the ontology it will infer that Student connect the CSModule and MathModule by the object Property

Studies. And Lecturer connects the CSModule and MathModule by the object Property Teaches.

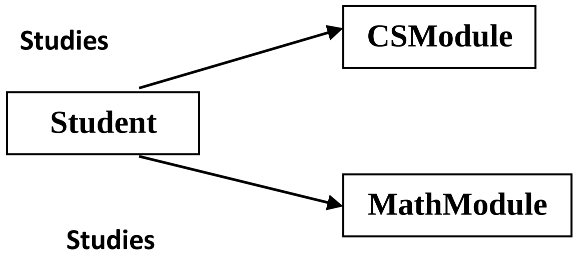


Figure 2.2: illustrates the fact of Student after reasoning.

Another core principle of a ontology is reusability. In order to share knowledge across various systems and to ensure interoperability of these systems, ontologies are often reused within other ontologies. Besides the simplification of knowledge exchange, ontology reuse tries to avoid duplicate work and reduces the work that is necessary to create a new ontology for a domain.

Many standards such as RDF, RDFS, or OWL have been published in the context of the W3C Semantic Web Activity by the World Wide Web Consortium (W3C). The Semantic Web is an approach to enrich the World Wide Web with machine-interpretable metadata using the technologies described in this section in order to allow better interoperability between Web pages and to ease knowledge sharing.

2.1.3 Web Ontology Language (OWL)

The Resource Description Framework (RDF) is a standard model for knowledge representation. It is specified in a set of recommendations by the World Wide Web Consortium (W3C).

In RDF, the term resources is used for instances. Each resource can have an arbitrary number of properties, i.e. attributes that associate literate values (e.g. numerical values, strings) to the resource or relations that link this resource to other resources. Resources and properties are expressed using statements (triples) which consist of three parts that are called subject, predicate, and object. To identify resources and properties, RDF uses URIs (Unified Resource Identifiers). In case a resource does not have an identifier, it is a blank node.

.

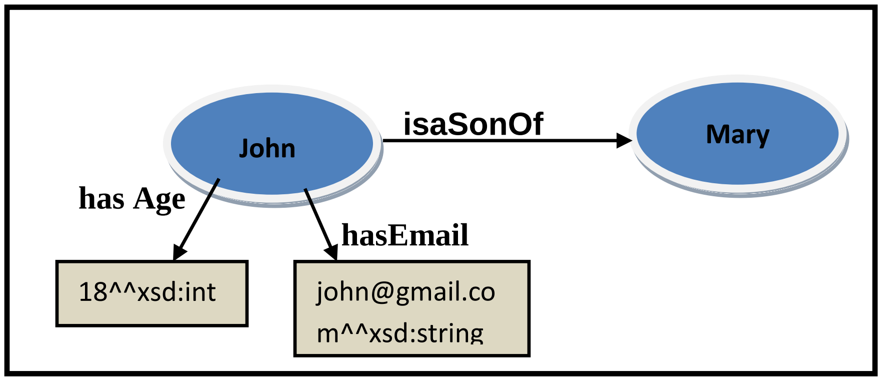


Figure 2.3: Example of a simple RDF model.

Figure 2.3 depicts a simple example for a piece of knowledge that John isaSonOf Mary here **isaSonOf** is a object Property which gives the relation between john and Mary. John has two data property **hasAge** and **hasEmail** which link two literal values to the resource: hasAge value of 18^^xsd:int and **hasEmail john@gmail.com**^^**xsd:string** type xsd:int and xsd:string of both literals is defined in XML Schema , one of the several XML schema languages available that define the structure of XML documents (Extensible Markup Language).

The complete URI of John is http://example.org/Jhon# and the URI of http://example.org/Jhon#. As in XML, substrings at the beginning of URIs may be replaced by prefixes to avoid frequent recurrences of the same strings. The part of the URIs replaced by the prefix is called a namespace which is used to group URIs for elements from the same source together; e.g. all concepts, properties, and individuals defined by weather phenomenon prediction have identifiers in the same namespace. For expressing the data that is represented by an RDF model, several serialization formats are available. The RDF recommendation is based on RDF/XML which maps the RDF model to

```
<?xml version="1.0"?>
<rdf:RDF xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:fam="http://example.org/Familyr#">
<rdf:Description rdf:about=" http://example.org/Family #John">
< fam:hasAge
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
18
</ fam:hasAge >
< fam:hasEmail
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
John@gmail.com
</ fam:hasEmail >
<fam:isSonOf  rdf:resource="http://example.org/ Family #Mary" />
</rdf:Description>
</rdf:RDF>
```

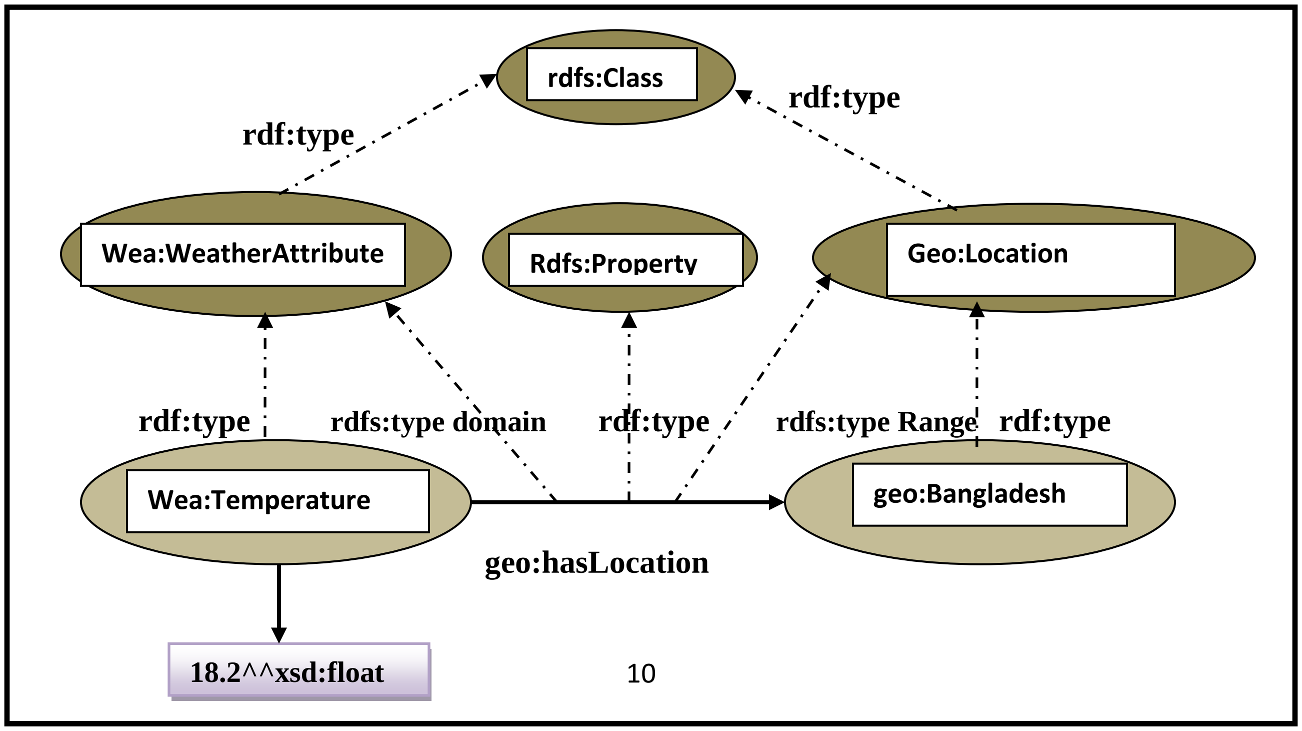
Listing 2.1: RDF example from Figure 2.3 encoded in RDF/XML syntax.

```
PREFIX fam:: <http://example.org/Family#> .
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> .
fam:John  fam:hasAge "18"^^xsd:int ;
fam:John  fam:hasEmail "John@gmail.com"^^xsd:int ;
fam:John  fam:isSonOf  fam:Mary.
```

Listing 2.2: RDF example from Figure 2.3 encoded in Turtle syntax.

an XML document . The representation of the above example in RDF/XML can be seen in Listing 2.1. As RDF/XML is a rather verbose format which may be difficult to read for humans, the N3 (Notation3) representation for RDF is available which was developed with human-readability in mind. Notation3 incorporates some syntax features that go beyond the expressive power of RDF. A subset of Notation3 named Turtle (Terse RDF Triple Language) is available that is limited to the features required to map RDF models .Listing 2.2 shows the above example in Turtle syntax.

RDF schema (RDFS) is a recommendation by the W3C that builds upon RDF. It introduces a set of concepts and properties adding features that go beyond the expressive power of RDF. All things described by RDF are instances of rdfs:Resource. Concepts – which are introduced by RDFS – are instances of rdfs:Class, and properties are instances of rdfs: Property. Other concepts introduced by RDFS are rdfs:Literal, rdfs:Datatype, and rdfs:XMLLiteral.



wea:hasTempValue

Figure 2.4: Example of a simple RDFS model.

The property rdfs:domain states that any resource that has a given property is an instance of one or more classes; the property rdfs:range states that the value of a property is an instance of one or more classes. rdf:type (often abbreviated with a) is used to state that a resource is an instance of a class. Using the property rdfs:subClassOf, hierarchies of classes can be constructed: C1 rdfs:subClassOf C2 states that any instance of C2 is also an instance of C1. rdfs:subPropertyOf is an equivalent that is used for declaring hierarchies of properties. Other properties defined by RDFS are rdfs:label and rdfs:comment.

Figure 2.3 shows the enriched by some elements that are introduced by RDFS. The data model introduces two classes, weather:WeatherAttribute and geo:Location. weather:Temperature and geo:Bangladesh are instances of weather: Weather Attribute and geo:Location, respectively. geo:hasLocation is a property with domain weather:Weather Attribute and range geo:Location.

RDFS comes with reasoning support e.g. in the above example, the statements can be removed without loss of knowledge.

```
wea:Temperature  rdf:type  wea:WeatherAttribute .
geo:Bangladesh  rdf:type  geo:Location .
```

A reasoner can deduce them from these statements:

```
weather:Temperature  geo:hasLocation  geo:Bangladesh .
geo:hasLocation rdfs:domain weather:WeatherAttribute .
geo:hasLocation rdfs:range geo:Location .
```

Many of the concepts and properties defined by RDFS are included in the Web Ontology Language (OWL), a more expressive ontology language than RDFS which is based on RDF and RDFS. OWL is developed by the OWL Working Group of the W3C. OWL was first published in July 2002 as a working draft and became a W3C recommendation in February 2004 ; the first working draft of OWL 2 was released in March 2009 and the W3C recommendation of OWL 2 in October 2009 with a second edition being finally released in December 2012. OWL 2 remains fully compatible to OWL, i.e. all OWL 1 ontologies are OWL 2 ontologies as well, with unchanged semantics.

Compared to RDFS, OWL introduces the following elements (among others which are omitted here):

1. Properties are instances of owl:ObjectProperty, owl:DatatypeProperty, or both; the property is termed object property or datatype property, respectively. An object property links an individual to another individual while a datatype property links an individual to a literal value.

2. The property owl:equivalentClass is used to state that two classes are equivalent while owl:allDisjointClasses states that there is no individual that is an instance of more than one class from the defined set of classes.

3. Some ontology languages include a Unique Name Assumption which states that two different names always refer to different entities in the world. OWL does not make this assumption, but provides the properties owl:sameAs and owl:differentFrom that are used to explicitly state that two individuals are the same individual or that two individuals can never be the same individual.

4. The properties owl:intersectionOf, owl:unionOf, and owl:complementOf can be used to describe complex classes in a notation borrowed from set theory, e.g. if there are two classes, Man and Woman, the class Person can be defined as the class union of them (if no other classes exist).

5. Using one of the properties owl:allValuesFrom, owl:someValuesFrom, and owl: hasValue, a class can be defined based on the values or classes of their properties.

6. The cardinality of properties per class can be limited using the properties owl:minCardinality, owl:maxCardinality, and owl:cardinality

7. Properties can have various characteristics: A property can be an inverse property of another property (owl:inverseOf) and two properties can be disjoint (owl:property DisjointWith). A property can be reflexive (i.e. it relates everything to itself, owl:ReflexiveProperty), irreflexive (no individual can be related to itself, owl:IrreflexiveProperty), functional (every individual can be linked to at most one other individual, owl:FunctionalProperty), or inverse functional (the inverse property is functional, owl:InverseFunctionalProperty).

Reasoning: Reasoning in OWL respects the Open World Assumption. If some statement cannot be inferred, it is not allowed to assume that this statement is false. Hence, reasoning in OWL is monotonic: Adding more information to a model cannot cause anything to become false that has previously known to be true, and vice versa. Queries on RDF, RDFS, and OWL models are often performed using SPARQL, a query language for RDF . Listing 2.3 shows an example for the use of SPARQL to query all temperature values known for Bangladesh in the RDFS model depicted in Figure 2.5. SWRL is a language for OWL which is used to express rules. SWRL can be used to define relations that may be difficult or impossible to define using OWL alone, e.g. due to the Open World Assumption. OWL ontologies are often designed using semantic editors like Protégé. Common reasoners for OWL include Pellet, RacerPro , FaCT++ , and HermiT.

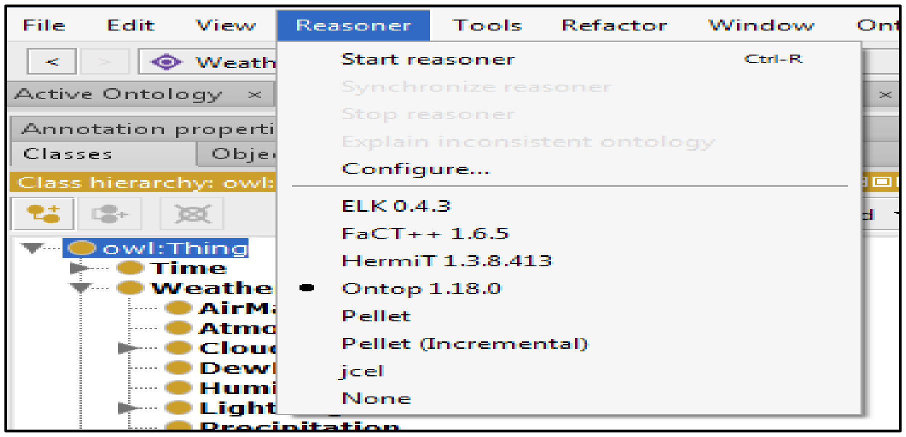


Figure 2.5: List of Reasoner implemented in Protégé

```
@prefix weather: <http://example.org/weather#>
@prefix geo: <http://example.org/geo#>
SELECT ?temperature
WHERE {
?state a weather:WeatherAttribute.
?state geo:Location geo:Bangladesh.
?state weather:hasTemperatureValue ?temperature.
}
```

Listing 2.3: SPARQL code to query all known temperature values in the model from Figure 2.4

2.2 Ontology-Based Data Access

Over the past 20 years we have moved from a world where most companies had one all-knowing self-contained central database to a world where companies buy and sell their data, interact with several data sources, and analyze patterns and statistics coming from all of them. The challenge is shifting from obtaining information to finding the right information. It has always been the case that information is power but today attention rather than information becomes the scarce resource, and those who can distinguish valuable information from background clutter gain power. To separate the wheat from the chaff, the companies need a comprehensive understanding of their data and the ability to cope with diversity in the data.

Since the mid 2000s, Ontology-Based Data Access (OBDA) has become a popular approach to tackling this problem. In OBDA, a conceptual layer is given in the form of an ontology that defines a shared vocabulary, models the domain, hides the structure of the data sources, and can enrich incomplete data with background knowledge. Then, queries are posed over this high-level conceptual view, and the users no longer need an understanding of the data sources, the relation between them, or the encoding of the data. Queries are translated by the OBDA system into queries over potentially very large (usually relational and federated) data sources. The ontology is connected to the data sources through a declarative specification given in terms of mappings that relate symbols in the ontology (classes and properties) to (SQL) views over data. The W3C standard R2RML was created with the goal of providing a language for the specification of mappings in the OBDA setting. The ontology together with the mappings exposes a virtual RDF graph, which can be queried using SPARQL, the standard query language in the Semantic Web community. These virtual RDF graphs can be materialized, generating RDF triples that can be used with RDF triple stores, or alternatively they can be kept virtual and queried only during query execution.

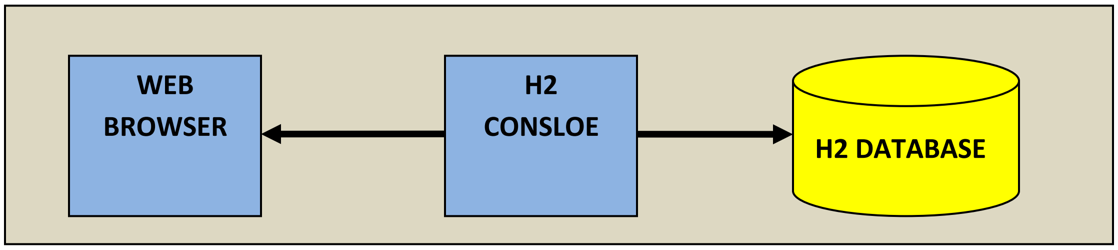
Here we show the Example for (Hospital Database)[7]. We consider a hospital database with a single table tbl_patient that contains information about lung cancer patients. The table has 4 attributes: the patient identifier (pid), his/her name, the type of cancer (tumor) and its stage. The lung cancer can be of two types: Non-Small Cell Lung Carcinoma (NSCLC) and Small Cell Lung Carcinoma (SCLC), which are encoded in the table by a boolean value type as follows: – false for NSCLC and true for SCLC. The stage of the cancer is encoded by a positive integer value stage as follows: – 1–6 for NSCLC stages I, II, III, IIIa, IIIb and IV, – 7–8 for SCLC stages Limited and Extensive. Finally, our sample table contains the following data: pid name type stage

pid	name	type	Stage
1	Mary	false	4
2	John	True	7

Suppose we need a simple piece of information from this database: “Give me the names of patients with a tumor of stage IIIa”. Even this simple query in this tiny database already presents some challenges, since to create the query and to understand and analyze the results we need to know how the information is encoded in the data. In the following sections we describe how to use the Ontop system to address this challenge by enhancing the database with a semantic layer.

2.3 Relational Database H2

H2 is a relational database management system written in Java. It can be embedded in Java applications or run in the client-server mode[14]. As this database is written in Java, it can run on many different platforms. The database is developed and tested on Windows 8 and Mac OS X using Java 7, but it also works in many other operating systems and using other Java runtime environments. All major operating systems (Windows XP, Windows Vista, Windows 7, Mac OS, Ubuntu,.) are supported.



A subset of the SQL (Structured Query Language) standard is supported. H2 database mainly support APIs are SQL and JDBC. However it also supports the PostgreSQL ODBC driver by acting like a PostgreSQL server. The database can be created both in-memory tables, as well as disk-based tables. All data are operating in transaction mode. Locking and multilevel concurrency controls and commit are implemented in H2 database. But no standard API for distributed transactions is implemented. The Security features of this database are used high level encryption such as SHA-256 and data using the AES or the Tiny Encryption Algorithm, XTEA. H2 Supports all kind of Operating System Environment. H2 currently supports three servers: a web server (for the H2 Console), a TCP server (for client/server connections) and an PG server (for PostgreSQL clients) The servers can be started in different ways, one is using the Server tool. Starting the server doesn't open a database - databases are opened as soon as a client connects.

2.4 Relational database vs. Resource description framework (RDF)

A relational database is a set of tables containing data fitted into predefined categories. Each table (which is sometimes called a *relation*) contains one or more data categories in columns. Each row contains a unique instance of data for the categories defined by the columns. When you create relationship between one or more table you need to make sure that number of row and columns should be equal. Every table shares at least one field with another table in one-to-one, one-to-many or many-to-many relationships. These relationships allow the database user to access the data in almost an unlimited number of ways and to combine the tables as building blocks to create complex and very large databases.

Resource description framework (RDF) on the other hand, is a set of triple concepts that is expressed as simple *subject–predicate–object.* A *triple* is also known as a "statement" and is the basic "fact" or asserted unit of knowledge in RDF. Multiple statements get combined together by matching the *subjects* or *objects* as "nodes" to one another (the *predicate*s act as connectors or "edges"). As these node-edge-node triple statements get aggregated, a network structure emerges, known as the RDF *graph*. RDF triples can be applied equally to all structured, semi-structured and unstructured content. By defining new types and predicates, it is possible to create more expressive vocabularies within RDF. This expressiveness enables RDF to define controlled vocabularies with exact semantics. In RDF everything is stored as statements with three pieces of information, you can add new predicates to the database as much as you want without changing any schema, and nothing will change except for how far you can reach in terms of complex query capacity. In other words, it's easy to grow your database. Following example gives more clear idea about the RDF.

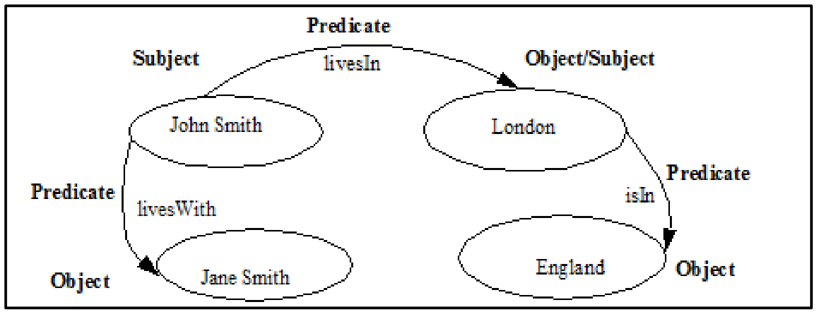


Figure 2.6: RDF graph

Here "John Smith" is the subject, "livesIn" is predicate and "London" is the object or property. In RDF/XML, each part is defined by a URI (Uniform Resource Identifier). Subject and predicate must be URI and object may be URI or literal type. One subject or object can be connected to other subject or object of RDF graph using Predicate. If one object is connected to another subject then object must be URI. In this way we can able connect the huge dataset in the form of subject, predicate and object. And this the main advantage of using RDF rather than Relational database. There are several query languages designed to query RDF databases (triplestores), RDQL, SPARQL, RQL, SeRQL, Versa, and more. The most popular language of them all is SPARQL, by far, which could be characterized as being something like a combination of SQL, Turtle, and a dynamic table of contents. Much like SQL has views and stored procedures, SPARQL has named graphs, which is itself a resource.

Chapter 3

Weather Data Source

In this chapter aims at compiling a set of weather data elements which are either necessary for providing useful data on exterior influences to weather phenomenon or which would add benefit to the data provided. Furthermore, possible sources are evaluated with respect to their suitability for the given context.

3.1 Weather information

In this project my aim is to determine the weather attribute that are associated with certain weather condition. In order to identify which attribute is required for the weather phenomenon ontology, it is necessary to define the scope that shall be covered by the ontology. When designing an ontology, requirements analysis is often centred around a set of competency questions. If the ontology is able to provide answers to all of these competency questions, its requirements are met.

1. What are the factor are associated with thunderstorm?
2. Which type cloud is required to form thunderstorm?
3. Which season does it frequently occurs.
4. What is sea temperature is required to form cyclone.
5. How low pressure is affected the cyclone.
6. What is the dropletSize of the drizzle rain?

3.2 Service data

This section presents the details of a number of popular weather services that are available over the Internet. In a first step, a number of aspects are identified that are relevant for the usage of a weather service in the context of weather phenomenon. There is a tremendous amount of services providing weather data over the Internet. But In weather phenomenon detection I have only follow few of them. These are

1. Bangladesh meteorology department,
2. AccuWeather and
3. open weather map

from this sites I gather weather information which I implement in the database to predict the factors related to the specific weather condition.

3.3 Google search engine

Most of the information gathers from various weather related web sites that I search in google search engine. Basically I got the knowledge about the weather attribute which are related to various weather condition from the Wikipedia. In Wikipedia the factors are briefly describe for the weather condition that associated for that condition. Portal: Weather is another web site where I take the information about the attribute.

Chapter 4
Methodology

4.1 Proposed System Architecture

The system we proposed in this project is built maintaining four different levels:

1. Ontology is created which stores database metadata information within the basic ontology structure. Ontology is a formal explicit description of concepts in a particular domain. Ontological model includes classes, data types, object properties, data properties, assertions, individual instances of a class and other semantics. An ontology together with a set of individual instances of classes constitutes a knowledge base. We propose a knowledge base for the purpose to analyze weather condition. The Weather phenomenon has different Attribute such as Temperature, Precipitation, and Humidity. To give them common platform to manipulated their data. We built an ontology which consists of the hierarchy of descriptors that describe both the Weather Attribute and Weather Conditions to describe the relation between them.

2. We have a H2 database that is used as a data source for those descriptor and attribute values that are fact of occur the Weather Condition. The database contains three tables-one for storing the data to describe the types of Weather Condition, and the Second Describe the Month of occurrences and their Duration and the third tables describe the Weather Attribute and their values. The descriptors and the attributes built in the ontology are now mapped to their corresponding values stored in the database. The database-to-ontology mapping is generally regarded as a case of data integration. The goal of data integration system is to provide a common interface to various data sources to enable users to focus on specifying what they want. In our proposed system, the database is the source schema and the ontology model is the target schema. Therefore, data integration or data mapping can be described as the problem of creating correspondences between sets of relational and ontological data. To define the problem of database and ontology mapping, taking our proposed system into account, we are given:

- An ontology expressed in RDF; the ontology contents is viewed as a set of triples.
- A relational database instance, whose contents are stored into tuples.

The objective of data mapping is to find mappings and create a set of correspondences relating ontological data – predicates in the ontology, and the relational data tuples.

3. Ontop data source manager is used to established connection with H2 database using JDBC. JDBC is a java database Connectivity is an application programming interface (API) for the programming language Java, which defines how a client may access a database.

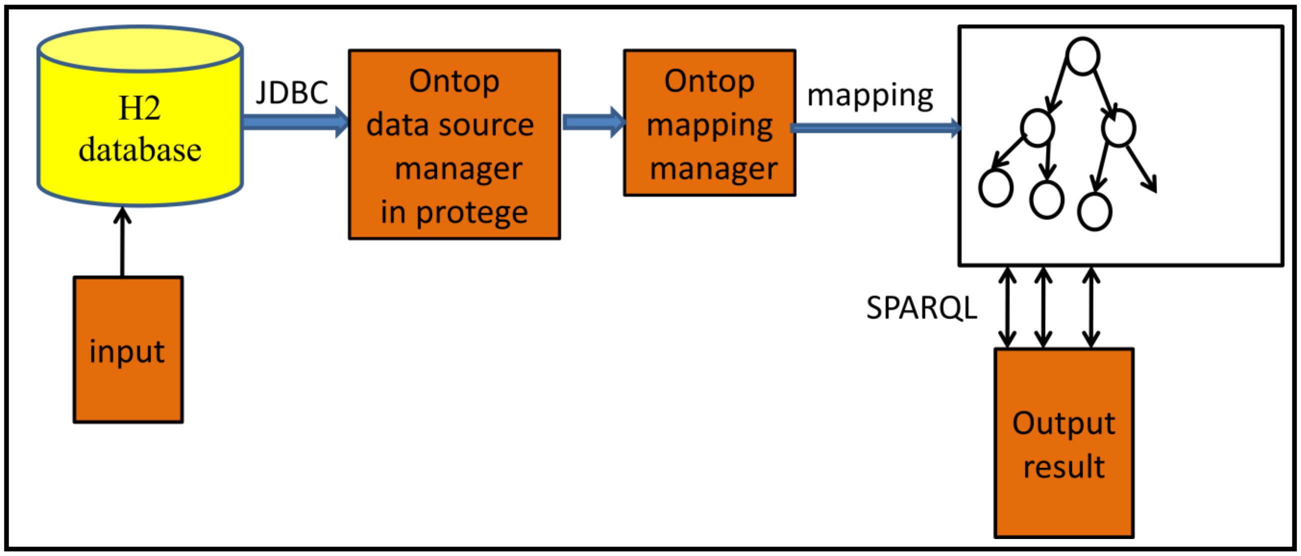


Figure 4.1: Flow diagram of the proposed system architecture

4. Ontop mapping manager is used where we mapped the data to our ontology. It has three parts; these are target, mapping id, and source. Target where we can convert the SQL data to RDF format. It also provides where the data will be mapped. Source is used where we should retrieve the data. Then SPARQL query is used to query over the complete ontology to execute the output.

4.2 Development of Ontology

There are numerous methodologies for building ontologies. In the article about their approach towards ontology design, Noy and McGuinness[2] state three fundamental rules of ontology design. Although the authors only apply them to their own approach, they hand out advice for many design decisions, regardless of which approach is used for the design of the ontology:

1) There is no one correct way to model a domain – there are always viable alternatives. The best solution almost always depends on the application that you have in mind and the extensions that you anticipate.

2) Ontology development is necessarily an iterative process.

3) Concepts in the ontology should be close to objects (physical or logical) and relationships in your domain of interest. These are most likely to be nouns (objects) or verbs (relationships) in sentences that describe your domain.

4.3 The ontology development approaches

- **Formality:** The ontology design process can reside on an informal level (the ontology and all artifacts created during development are described using natural language, in tables, and in diagrams), a formal level (all aspects of the ontology are described using the logical model of the ontology language the ontology is intended to be implemented in), or anything in between.

- **Level of detail**: The level of detail of the description of the design process can range from giving just an overview to a level describing every step in a very detailed manner.

- **Documentation:** One methodology may enforce the creation of documentation while others may delegate the decisions about how the documentation is structured and what is documented to the ontology designer. This may lead to missing, inaccurate, or incomplete documentation.

In their paper Ontology Development 101: A Guide to Creating Your First Ontology [2], Noy and McGuiness present an informal and rather intuitive approach for building ontologies from scratch. It is geared towards people without or with little prior knowledge about how to design ontology and qualifies for demonstrating the essence of ontologies

The approach is divided into a set of steps:
1. The domain and scope of the ontology are determined. The preferred way for this is to formulate competency questions the ontology should be able to answer.

2. Existing ontologies are considered to be reused to avoid doing work that has already been done and to simplify interoperability with other ontologies.

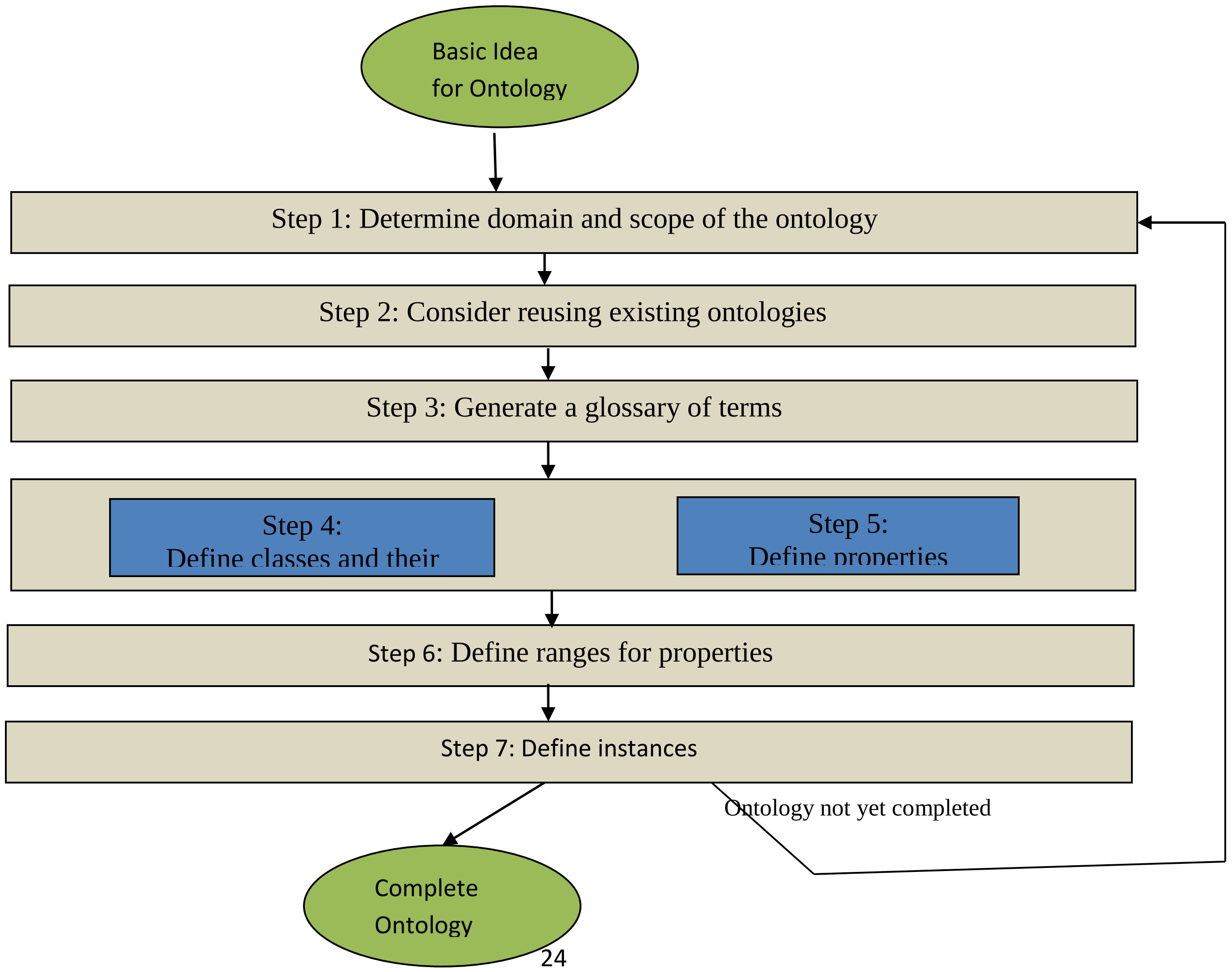
3. Important terms in the ontology are enumerated, i.e. a glossary of terms is built.

4. From the glossary in the previous step, all terms that are classes are identified. They are then related to each other in order to create a class hierarchy.

5. The next step iterates over all classes and tries to identify terms from the glossary which are properties of the classes.

6. Then, for the properties the ranges of possible values are specified.

7. Finally, instances from the glossary are selected and added to the ontolog



Basic Idea for Ontology
Step 1: Determine domain and scope of the ontology
Step 2: Consider reusing existing ontologies
Step 3: Generate a glossary of terms
Step 4: Define classes and their
Step 5: Define properties
Step 6: Define ranges for properties
Step 7: Define instances
Ontology not yet completed
Complete Ontology

Figure 4.2: The workflow proposed by Ontology

4.3.1 Ontology development process and life cycle

The ontology development process used by methodology divides the process into the following activities that need to be performed:

- **Planning**: This step involves creating a plan regarding which tasks need to be done and how they are arranged. As methodology already proposes such a plan, this step is omitted when following methodology to design ontology.
- **Specification:** The purpose, intended uses, and end-users of the planned ontology are specified in an Ontologies Requirements Specification Document.
- **Knowledge acquisition:** Knowledge about the ontology's domain is acquired.
- **Conceptualization:** The knowledge previously acquired is conceptualized into a model that describes the problem that shall be solved by the ontology and how the ontology is intended to solve it.
- **Formalization:** This conceptual model is then formalized.
- **Integration:** As ontologies are built to be reused, as many existing ontologies as possible are to be integrated into the new ontology.
- **Implementation:** The ontology is then implemented using a formal language.
- **Evaluation:** Throughout the process of building the ontology, it is continuously evaluated in order to ensure it meets the previously specified requirements.
- **Documentation:** The ontology and all documents belonging to it must be well documented.
- **Maintenance:** It may be necessary to apply modifications throughout the lifetime of the ontology.

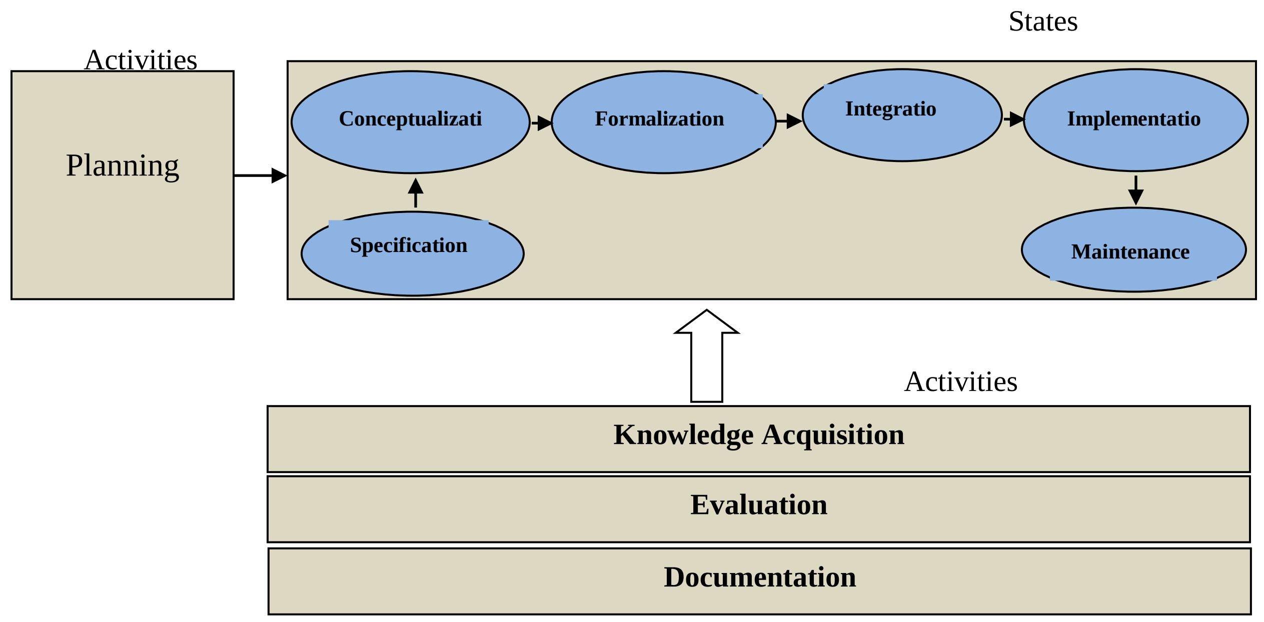


Figure 4.3: States and activities in the life cycle of an ontology according methonology

These activities – are arranged into the step of planning that must be performed at the very beginning of development, a set of stages (consisting of Specification, conceptualization, formalization, integration, implementation, and maintenance) which the ontology moves through during its creation, and some activities (knowledge acquisition, documentation, and evaluation) that are performed throughout the whole development process in parallel to the stages.

4.3.2 The methodology approach

This section describes methodology as a well-defined approach to perform all activities mentioned above.

Specification

Methodology defines a precise approach for the development of an ontology. It specifies certain activities that need to be performed, how these activities are performed, and in which order. Thus, the activity of planning is completed by specifying methodology itself and the ontology developer are exempted there from. Hence, the first step of developing ontology from scratch is specification

During specification, an ontology requirements specification document is generated. This document is written in natural language using a set of intermediate representations or using competency questions. It should include

- the name and the purpose of the ontology, its scope, its intended uses, and possible end users,
- a list of functional requirements (describing the intended functionality of the ontology) and non-functional requirements (describing all intended properties of the ontology not directly related to its functionality), and
- a list of terms that specifies the scope of the ontology.

Knowledge Acquisition

Most of knowledge acquisition is done simultaneously with the specification phase. It is one of the most important activities and needs to be performed thoroughly as most other activities heavily depend on it. Sources of knowledge are experts, books, handbooks, figures, tables, and even other ontologies. Knowledge is collected using techniques such as brainstorming, interviews, formal and informal analysis of texts, and knowledge acquisition tools.

Conceptualization

The state of conceptualization consists of several tasks. Again the tasks are performed in a sequential manner. However, as methodology uses an evolutionary process model, the steps are performed numerous times.

Name	Synonyms	Acronyms	Description	Type
-------	----------	---------	---------	--------

Table 4.2: Template for the glossary of terms as proposed by methodology of ontology.

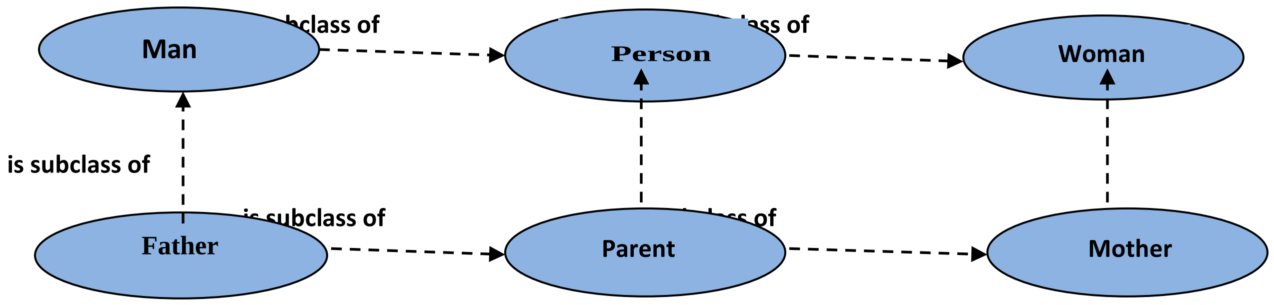


Figure 4.4: Example of a concept-classification tree as proposed by methodology.

Task 1: Glossary of Terms At first, a Glossary of Terms is built. This glossary includes all the relevant terms of the domain (concepts, instances, attributes, relations etc.). It can be built as a table having the columns name, synonyms, acronyms, description (for a natural language description of the term), and type (specifying whether the term is a concept, an instance, an attribute, a relation etc.). Table 4.2 shows a template for the glossary of terms.

Task 2: Concept Taxonomies Once the glossary of terms contains a sizeable number of concepts; these ontologies are arranged in one or more taxonomies that define the concept hierarchy.

4.4 Methodology proposes the use of four taxonomic relations

1. **Subclass-Of:** If a concept B is a Subclass-Of a concept A, every instance of B is also an instance of A.

2. **Disjoint-Decomposition:** A Disjoint-Decomposition of a concept C is a set of subclasses of C such that an instance of one of these subclasses can never be a subclass of another of these subclasses, while an instance of C is not necessarily an instance of one of its subclasses.

3. **Exhaustive-Decomposition:** An Exhaustive-Decomposition of a concept C is a set of subclasses of C such that every instance of C is an instance of at least one of its subclasses.

4. **Partition:** A Partition of C is a set of subclasses of C such that every instance of C is an instance of exactly one of its subclasses.

The concept taxonomies are visualized in concept-classification trees which are diagrams that depict the concepts and their taxonomic relations. See Figure 3.9 for an example of a concept-classification tree. In case the ontology contains a large number of concepts, the tree may be split into several diagrams in order to keep the trees clear.

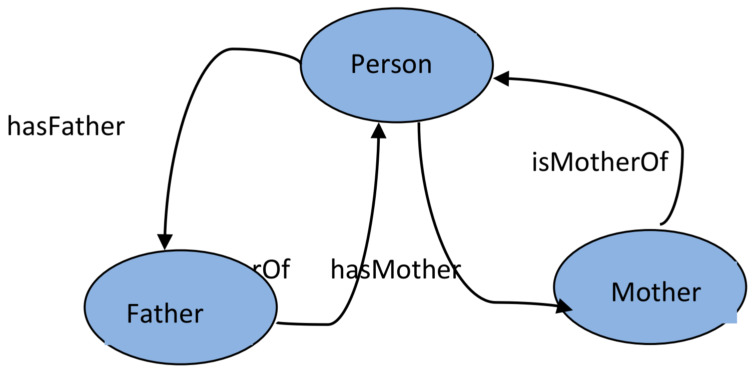


Figure 4.5: Example of a binary relations diagram as proposed by methodology.

Name	Instances	Relations	Class attributes	Instance attributes
------------	-------------	------------	--------------	--------------

Table 4.3: Template for the concept dictionary as proposed by methodology.

Relation name	Source concept	Target concept	Maximum source cardinality	Inverse relation
------------	-------------	------------	--------------	--------------

Table 4.4: Template for the binary relations table as proposed by methodology

Task 8: Constants In this step, the constants table is created that specifies details about all the constants listed in the glossary of terms. Each constant is specified by its name, its value type, its value, the measurement unit for numerical constants, and the attributes that can be inferred using the constant. Table 4.3 shows a template for the constants table.

Task 9: Formal axioms In this task, the ontology designer must determine whether the ontology contains formal axioms. In case it contains any, these axioms must be defined precisely in a formal axioms table. For each axiom, this table specifies the name, a description in natural language, the logical expression that formally describes the axiom in first-order logic (or the logic the ontology language intended to use is based upon), and all concepts, attributes, relations, and variables that are referred to in the logical expression. Table 4.4 presents a template for the formal axioms table.

Axiom name	Description	Expression	Referred concepts	Referred attributes	Referred relations	Referred variables
--------	--------	-------	--------	---------	----------	-------

Table 4.5: Template for the formal axioms table as proposed by methodology

Instance name	Concept name	Attribute	Value(s)
----------	-------	-----------	------------

Table 4.6: Template for the instants table as proposed by methodology

Task 10: Rules Similar to the task of identifying and describing formal axioms within the ontology, in this step the ontology designer must determine whether the ontology contains any rules. If it contains any, a rules table must be built to precisely describe all rules and their properties: Their names, their descriptions, expressions in first-order logic, and all concepts, attributes, relations, and variables involved. In contrast to formal axioms, the expression of rules always has the form if <conditions> then <consequent>; hereby <conditions> is a conjunction of atoms while <consequent> is a single atom.
For the rules table, the template for the formal axioms table (see Table 4.5) can be reused.

Task 11: Instances Within an ontology, a set of instances may be predefined. This task involves listing these individuals, again in tabular manner. The columns of this instances table are the name of the instance, the name of the concept, and the entire instance's attributes together with their respective values. Table 4.6 depicts a template for the instances table.

Formalization

Formalization is the transition from the informal description of the tables and diagrams in the previous step of conceptualization into the chosen ontology language, e.g. OWL. As this is tightly coupled with the implementation of the ontology, this is a task which is often not performed separately.

Integration

As ontologies are built for reuse and the wheel shall not be reinvented during the creation of a new ontology, the ontology designer searches for existing ontologies. The goal is to import ontologies that already define terms that are part of the ontology currently being developed.

Implementation

The task of implementing the ontology in an ontology language requires an environment that supports the ontologies selected in the integration step. Features that should be provided by such an environment are

- a lexical and syntactic analyzer to guarantee the absence of lexical and syntactic errors,
- an editor for adding, modifying, and removing definitions,
- a browser for inspecting the library of ontologies and their definitions,
- a searcher for looking for the most appropriate definitions,
- evaluators for detecting incompleteness, inconsistencies, and redundant knowledge, and
- an automatic maintainer for managing the inclusion, removal, or modification of existing definitions. In the case of the Weather Phenomenon ontology, an OWL ontology is created using Protégé together with the ontop reasoner.

Evaluation

During evaluation, verification takes place whether all artifacts that have yet been created or updated in the previous steps satisfy the requirements that have been initially defined. Evaluation is not an activity which is performed at the very end of the development process; instead, evaluation takes place whenever an artifact (a diagram, a table, or the implementation of the ontology) is created or updated in order to ensure that mistakes are found as soon as possible.
The completed ontology must fulfill all functional and non-functional requirements listed in the ontology requirements specification document. In case of a mismatch, the ontology traverses the activities of the life cycle (conceptualization, formalization, integration, implementation, and evaluation) once more.

Documentation

During the steps described above, a set of documents is compiled. If generated properly and accurately, these documents describe every detail of the ontology. Using this approach, methodology forces the ontology designer to document throughout the development process. Any problems that come with incomplete or wrong documentation are avoided. Hence, in methodology, documentation is an activity that is not performed explicitly. Once the development process has finished, both the ontology and its documentation are ready to use.

Maintenance

At any time in the future, changes to the ontology may become necessary. A modification of the ontology's requirements may be one reason; inaccurateness that occurred during the ontology development process may be another reason. Whenever a change is necessary, the ontology again cycles the states of specification, conceptualization, formalization, integration, and implementation repeatedly until all requirements are met and all artifacts generated in these states correspond to each other. Knowledge acquisition and evaluation are also again performed throughout all of these states.

4.5 Architecture of ODBA

The ODBC architecture consists of four components, as described in the following list.

- **Application programming interface (API)** Calls ODBC functions to connect to a data source, send and receive data, and disconnect.
- **Driver manager** Provides information to an application (such as a list of available data sources), loads drivers dynamically as they are needed, and provides argument and state transition checking.
- **Driver** Processes ODBC function calls and manages all exchanges between an application and a specific relational database. If necessary, the driver may translate the standard SQL syntax into the native SQL of the target data source.
- **Data source** Comprises the data and its associated database engine.

Your application uses the ODBC API to connect to a data source, submit SQL statements, fetch data, and disconnect. A driver manager sits between the application and the ODBC drivers, decides which driver to load, and manages communications as driver functions are called. Finally, the drivers implement the functions of the ODBC API for the particular database. The following drawing shows how these functions interact.

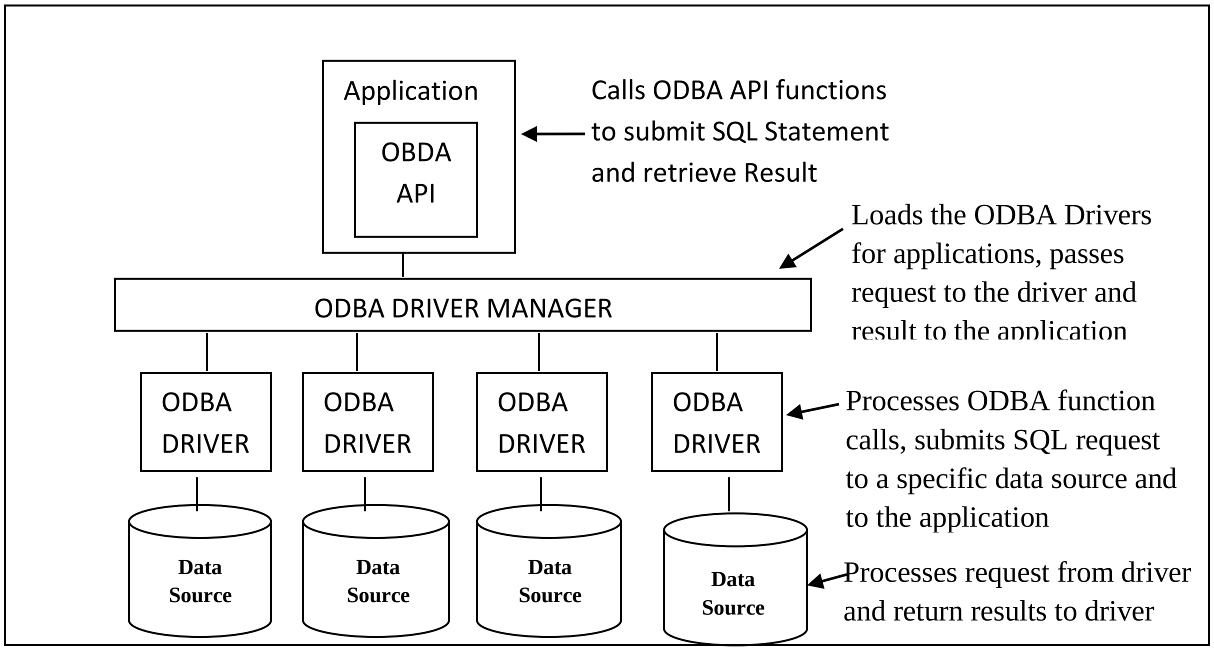


Figure 4.6: Architecture of Open database Access

What the ODBC architecture means to your application is that you can access different ODBC data sources, in different locations, using the same function calls available in the ODBC API. Once you have working code to access one relational data source, the code is easily extended to access other data sources.

Chapter 5
Implementation

The previous chapters cover all topics that require discussion before being able to build a new ontology from scratch: Chapter 1 discusses introduction and Problem Statement of Chapter 2 describes the Existing Work that are necessary to build this Project and Chapter 3 give an overflow diagram about the ODBA Architecture; Ontology Workflow and LifeCycle and System Architecture.

Based on these insights, this chapter describes the process of implementation the Weather Phenomenon in details. The implementation has been done in a Windows environment. Protégé version 5.1.0 has been used to build the ontology. H2 database engine is used as the data source for storing all the attribute values. We have used Fuseki-server as the SPARQL endpoint. The hierarchy of the ontology is shown in Figure1

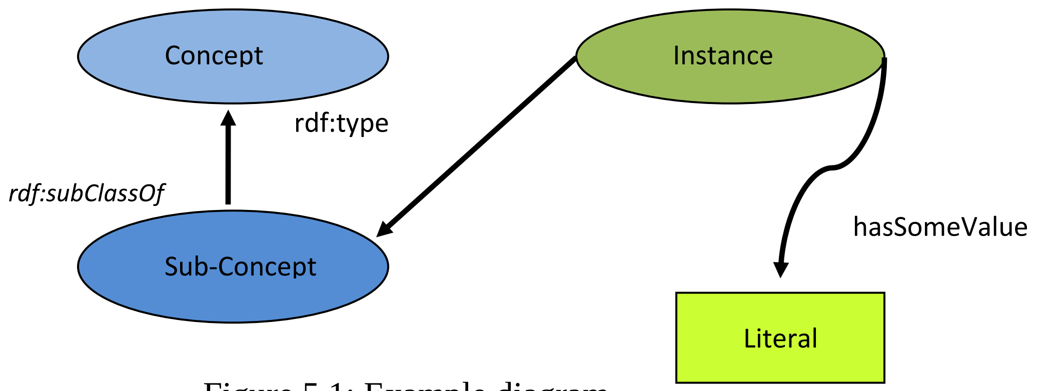


Figure 5.1: Example diagram.

5.1 Conventions

Every ontology should stick to a set of naming conventions that are explicitly stated the conventions for Weather Phenomenon Analysis are as follows:

- Two concepts, instances and/or properties may not have the same identifier as this required by OWL and avoids confusion.

- Two identifiers may not use names that only differ in their capitalization. Using both WEATHER ATTRIBUTE and weather attribute in the name namespace is possible in OWL, but lead to confusion.

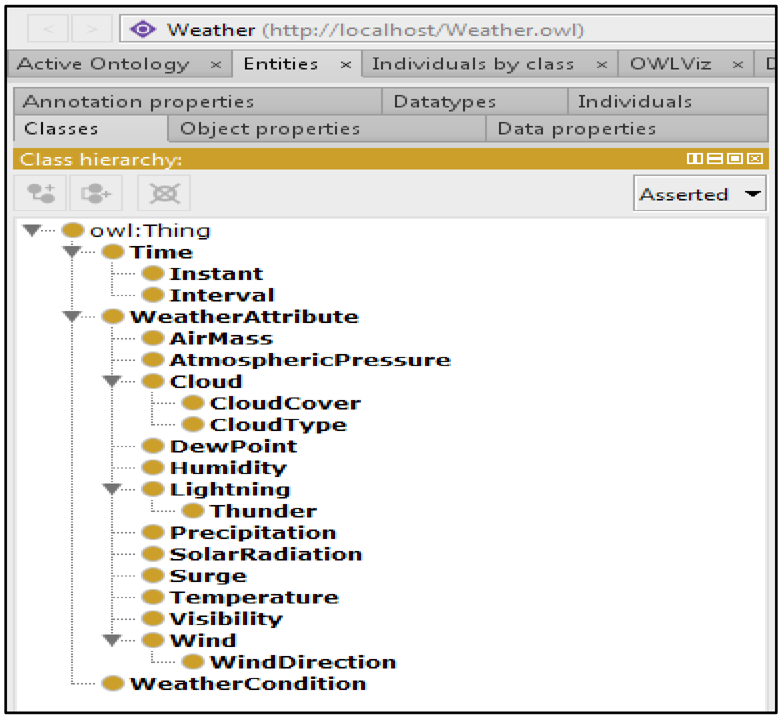
- Identifiers may only consist of upper and lower case **ASCII** letters (A to Z and a to z), numerical digits from 0 to 9 and spaces, i.e. all identifiers must match the regular expression^[A-Z, a-z, 0-9]+$.

- Concepts have an identifier that is in singular case and starts with an upper case letter. Typically a concept's identifier is a noun, e.g. Weather Attribute or Weather Condition.

- Properties have an identifier that starts with a lower case letter and starts with the prefix has or belongs to, followed by the name of the concept which is the property's range. The inverse property of a property having an identifier starting with has an identifier starting with belongs to, followed by the inverse property's range. As an alternative to the prefix belongs to, the prefix is in conjunction with the suffix of and the inverse property's domain may be used.

5.2 Create the ontology in Protégé

In this section, we propose a system for developing the knowledge base for weather phenomenon prediction. We have built an ontology to show the hierarchy of the weather attribute, time and weather condition and the attributes that have impact on the weather condition and the relationship between them. Protégé has been used as the ontology editor. The hierarchy is shown in figure 5.2.



```
owl:Thing
    Time
        Instant
        Interval
    WeatherAttribute
        AirMass
        AtmosphericPressure
        Cloud
            CloudCover
            CloudType
        DewPoint
        Humidity
        Lightning
            Thunder
        Precipitation
        SolarRadiation
        Surge
        Temperature
        Visibility
        Wind
            WindDirection
    WeatherCondition
```

Figure 5.2: Hierarchy for weather phenomenon prediction

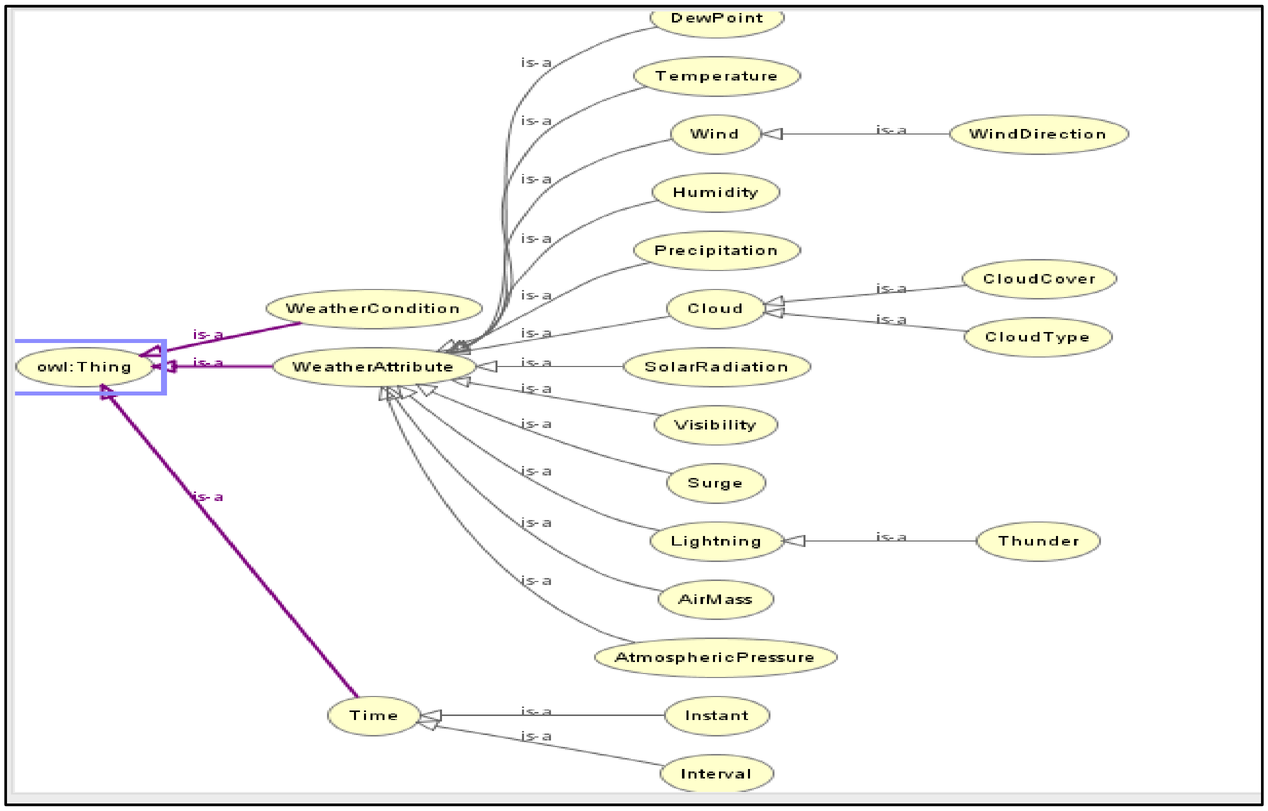


Figure 5.3: graphical View of The Hierarch of the Ontology

5.2.1 Top-Level Concept

Top-level Concept means the concept that do not have a superclass except owl:Thing. In this level I have three top-level concepts for the prediction of weather phenomenon. They are weather attribute, time and weather condition. Figure 5.4 show the top-level concepts

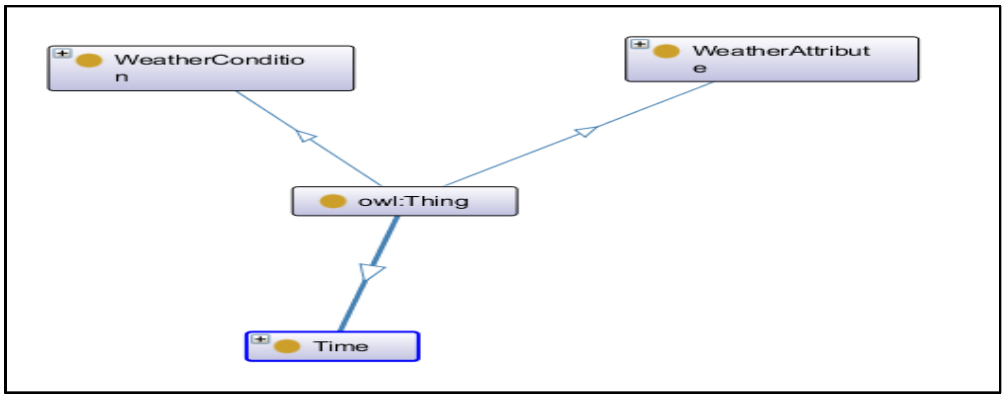


Figure 5.4: Super-Class-classification tree

5.2.2 Sub-concept information

Sub-concept Weather Attribute

In the ontology, weather elements are represented by concepts that are sub-concepts of Weather Attribute, e.g. there is a sub-concept **Temperature** for measurements of temperature, or **Humidity** for measurements of relative humidity. **Wind** for measurement of Wind Speed. **Precipitation** for the measurement of the Precipitation Intensity.

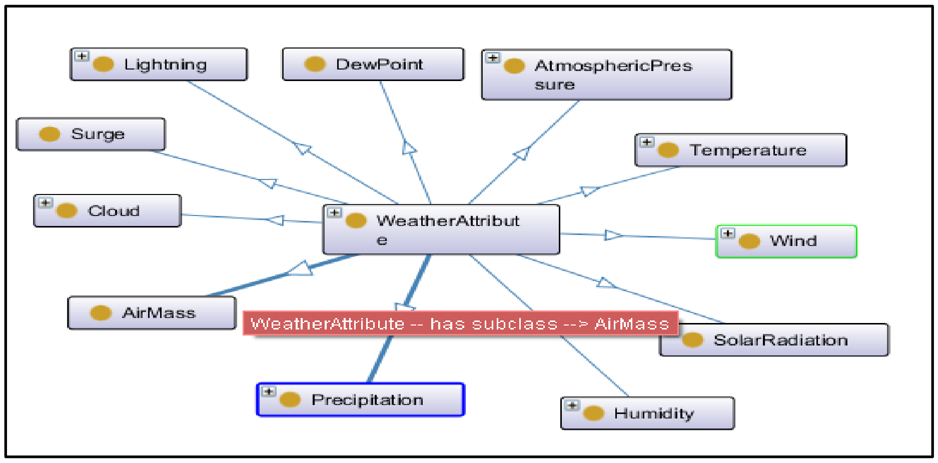


Figure 5.5: Sub-Concept-classification tree for Weather Attribute

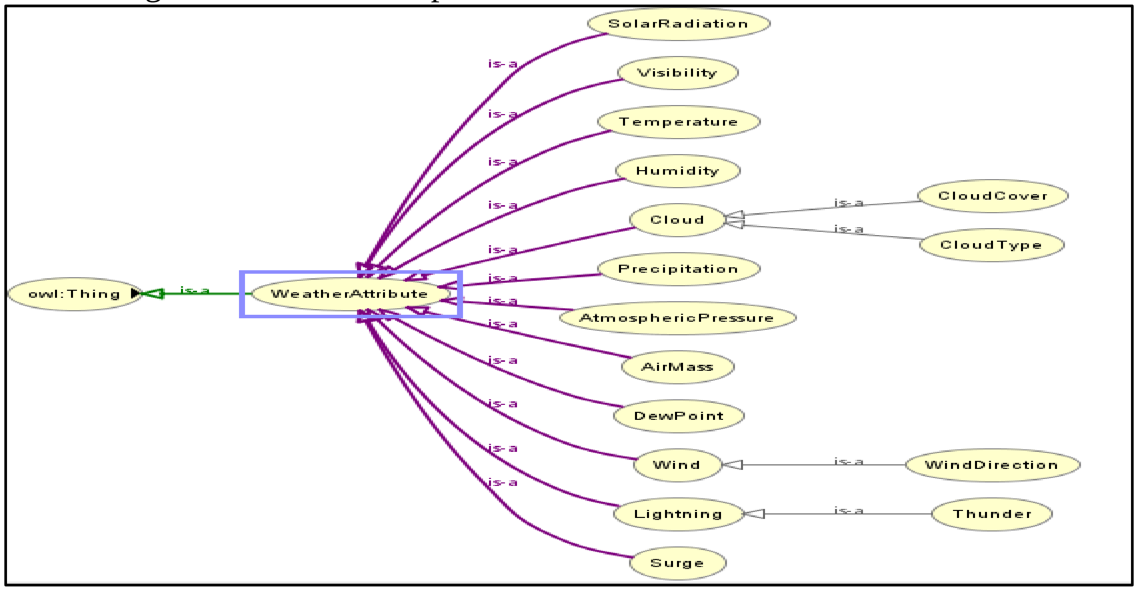


Figure 5.6: Hierarchy of Weather Attribute

Sub-Concept of Time

In the Ontology, Time and Date related information are represented by concepts that are sub-concepts of Time, e.g. there is a sub-concept **Interval** to describe the duration related information and Other sub-concept is Instant which represent the day, Month and Year related Information.

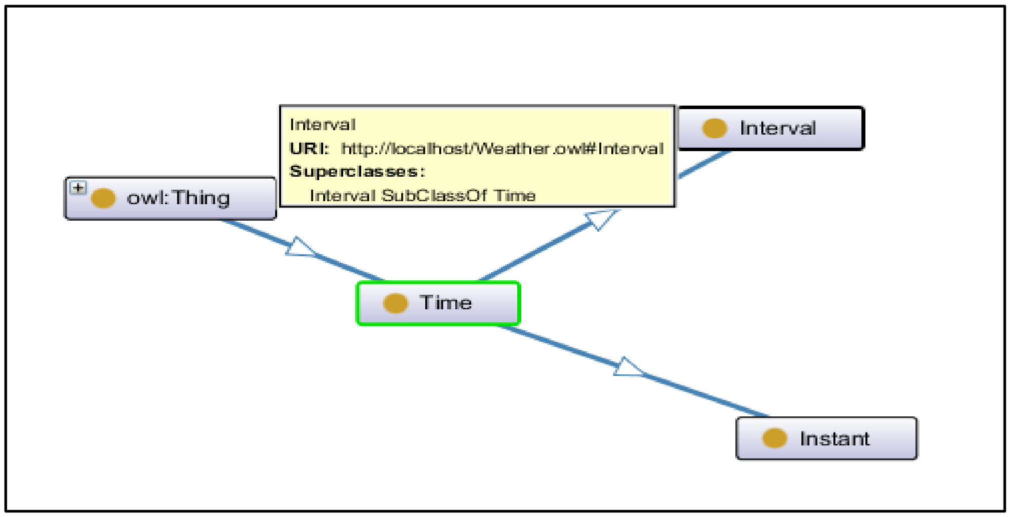


Figure 5.7: Sub-Concept-classification tree for Time.

Sub-Concept of Weather condition

A Weather condition does not have any sub-concepts.

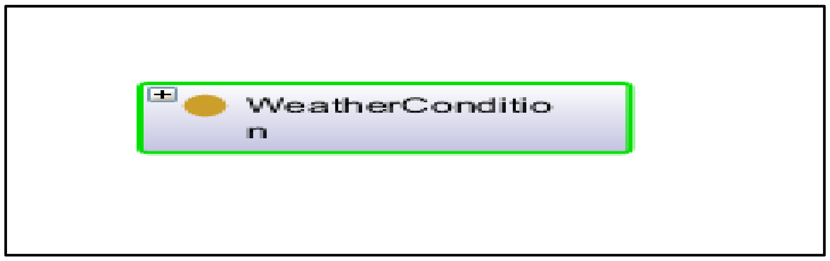
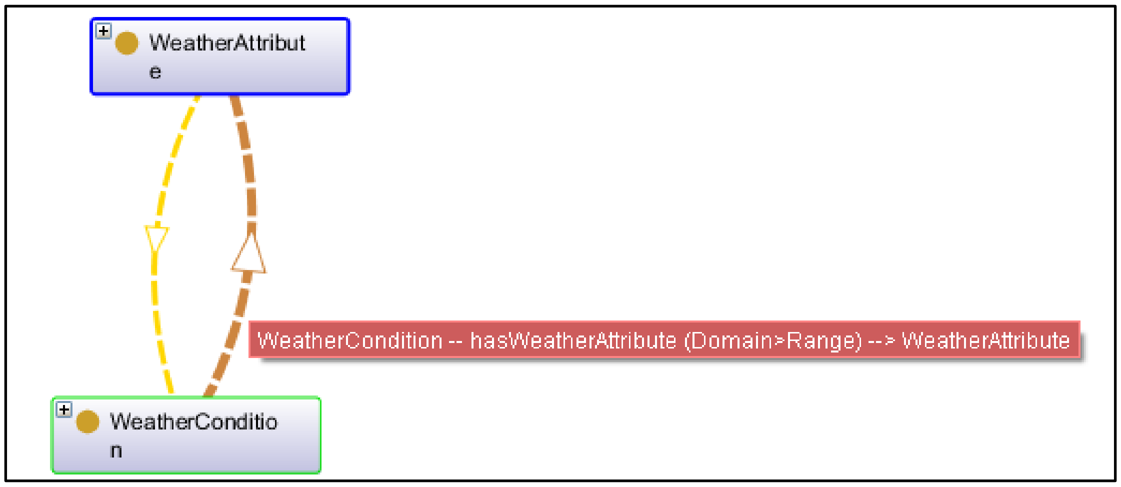


Figure 5.8: Sub-Concept-classification tree for Weather Condition

5.2.3 Object Property

Instances of the concepts are associated to each other with binary relations, which are:

- hasWeatherAttribute and belongsWeatherCondition which connect instances of Weather Attribute and WeatherCondition. And hasWeatherAttribute inverse of belongsToCondition. hasWeatherAttribute has domain:WeatherCondition and range:WeatherAttribute means this object property linked the weather condition class to weather attribute class.


WeatherAttribut e
WeatherCondition -- hasWeatherAttribute (Domain>Range) --> WeatherAttribute
WeatherConditio n
Figure 5.9: Relation between WeatherAttribute and WeatherCondition

- hasTimeAndDate which connects instances of WeatherCondition and Time. This object has domain: weather condition and range: Time. This means that weather condition class linked with time class by this property.

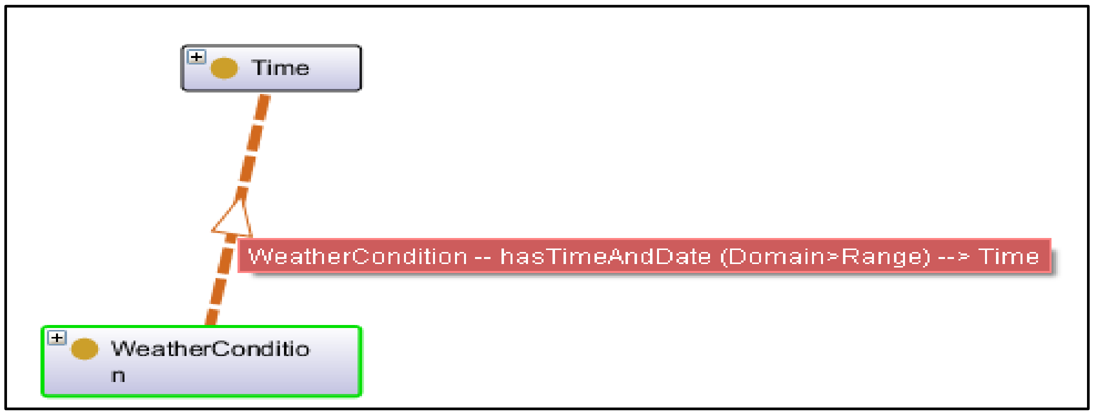
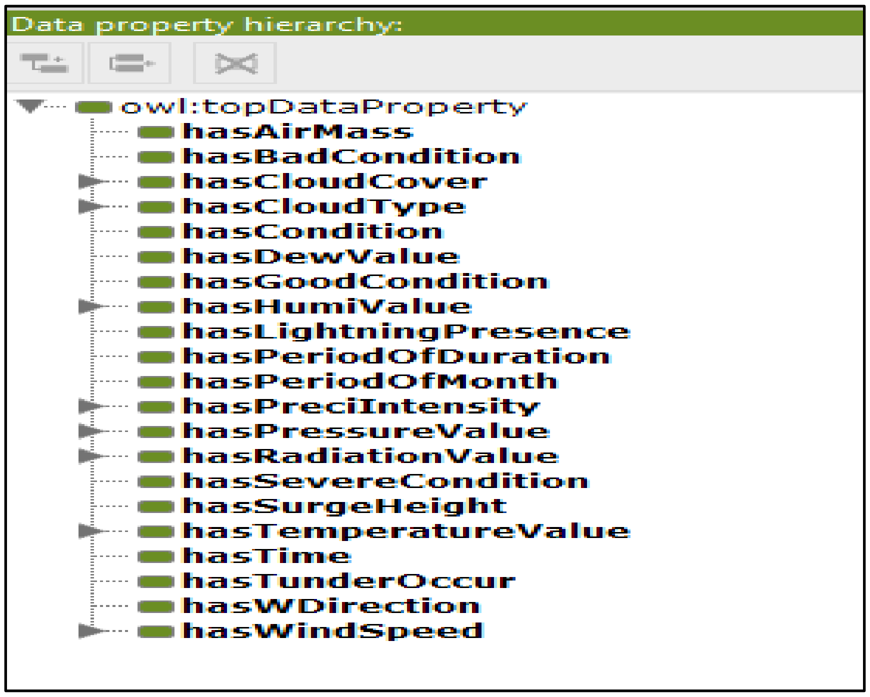


Figure 5.10: Relation between WeatherCondition and Time

5.2.4 Data Property

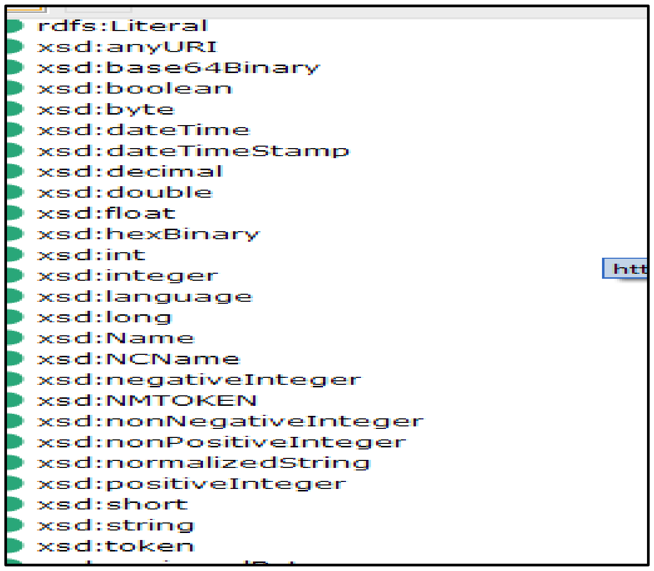
Data Property is used to define the Literal Values which is a granularity of the Concept tree. Data property connects the individuals to its literal value. There is several data type to define the value that what type value is it? Figure 5.10 shows the data property. And Figure 5.11 shows the data type.



Data property hierarchy:

```
owl:topDataProperty
    hasAirMass
    hasBadCondition
    hasCloudCover
    hasCloudType
    hasCondition
    hasDewValue
    hasGoodCondition
    hasHumiValue
    hasLightningPresence
    hasPeriodOfDuration
    hasPeriodOfMonth
    hasPreciIntensity
    hasPressureValue
    hasRadiationValue
    hasSevereCondition
    hasSurgeHeight
    hasTemperatureValue
    hasTime
    hasTunderOccur
    hasWDirection
    hasWindSpeed
```

Figure 5.11: Data property for Individual Humi/68



```
rdfs:Literal
xsd:anyURI
xsd:base64Binary
xsd:boolean
xsd:byte
xsd:dateTime
xsd:dateTimeStamp
xsd:decimal
xsd:double
xsd:float
xsd:hexBinary
xsd:int
xsd:integer
xsd:language
xsd:long
xsd:Name
xsd:NCName
xsd:negativeInteger
xsd:NMTOKEN
xsd:nonNegativeInteger
xsd:nonPositiveInteger
xsd:normalizedString
xsd:positiveInteger
xsd:short
xsd:string
xsd:token
```

Figure 5.12: Data type

5.2.5 Individuals

Individuals of Temperature

For all weather elements except **Dew point,** categories are introduced in order to allow easy differentiation of weather observations by their respective measurement values. In case of Temperature, the Individuals differ from each other by the observed temperature values. The Individuals of Temperature are Temp/1 (for an observed temperature value of GreaterThan 37°C), Temp/2 (at least 26°C and less than 37 °C), Temp/3 (at least 16°C and less than 25°C), Temp/4 (at least 1°C and at most 15°C), Temp/5 (LessThan 0°C), and Temp/6 (Sea Surface temperature Greater than 26°C).

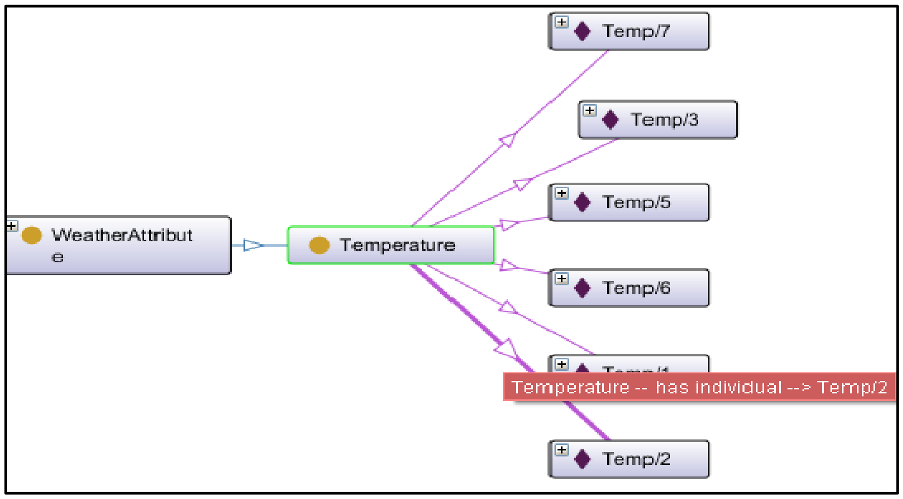


Figure 5.13: Individuals-classification tree for Temperature.

Individuals of Humidity

Humidity is the amount of water vapor present in the air. The value of different humidity is representing according to their Individuals. The individuals connect to value by date Property "hasHumiValue" The value are shown in below:

The Individuals of Humidity are Humi/76 (Less than 40 %), Humi/70 (at least 40 % and less than 50 %), Humi/73 (at least 81 % and less than 89 %), Humi/69 (at least 50 % and less than 70 %), Humi/68 (at least 71 % and less than 81 %), Humi/67 (Greater than 90%).

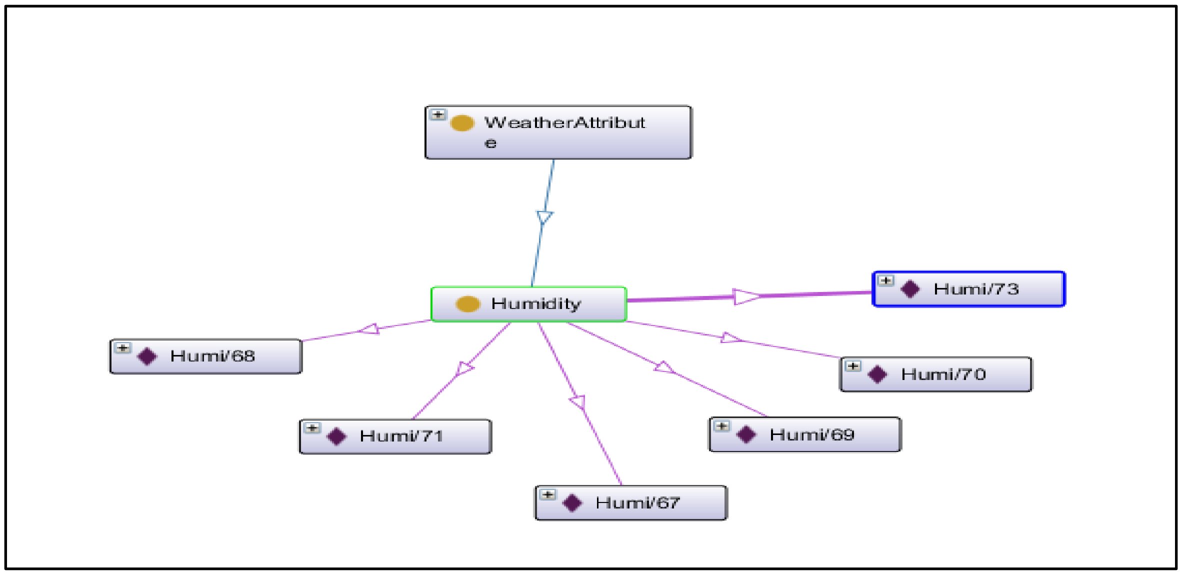


Figure 5.14: Individuals-classification tree for Humidity.

Individuals of Cloud Cover and Cloud Type

Cloud cover refers to the fraction of the sky obscured by clouds when observed from a particular location. Okta is the usual unit of measurement of the cloud cover The value of different Cloud Cover and Cloud Type is representing according to their Individuals. The individuals connect to value by date Property “hasCloudCover” The value are shown in below:

The Individuals of Cloud Cover are CloudCover /49 (Clear Sky), CloudCover /50 (Partly Cloud), CloudCover /51 (Mostly Cloud), CloudCover /52 (OverCast), CloudCover /53 (UnKnownCloud).

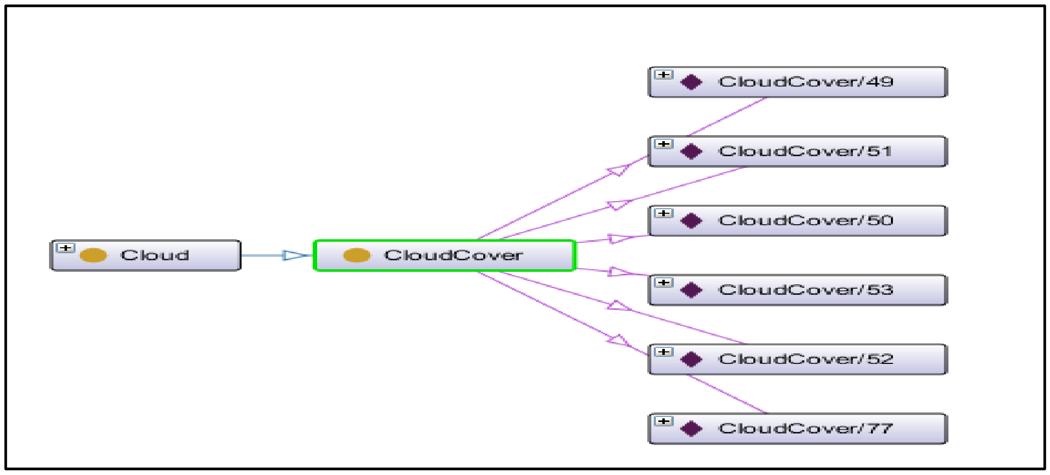
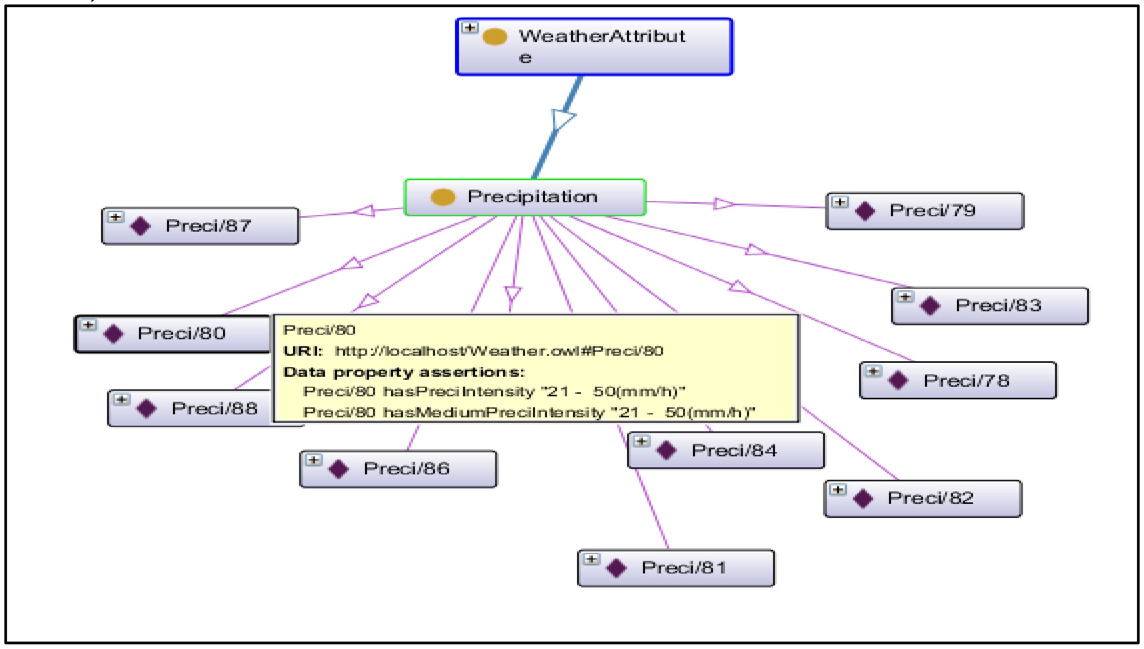


Figure 5.15: Individuals-classification tree for Cloud Cover.

Individuals of Precipitation

water that falls to the ground as rain, snow, etc. Precipitation is measure in Milimeters per hours. The individuals connect to value by date Property “hasPreciIntensity” The value are shown in below:

The Individuals of Precipitation are Preci /82 (0 mm/h), Preci /84 (1-5 mm/h), Preci /81 (6-20 mm/h), Preci /80 (21-50 mm/h), Preci /79 (51-100 mm/h), Preci /78 (greater than 100 mm/h)



Preci/80
URI: http://localhost/Weather.owl#Preci/80
Data property assertions:
Preci/80 hasPreciIntensity "21 - 50(mm/h)"
Preci/80 hasMediumPreciIntensity "21 - 50(mm/h)"

Figure 5.16: Individuals-classification tree for Precipitation.

5.3 H2 Database

We have used H2 database engine as the data source for storing all the weather related data . In H2 database I have Created a three table. These are TBL_WEATHERFORECAST, InTBL_WEATHERCONDITON, TBL_TIMEANDDATE. Description of each the table is describe in below.

In TBL_WEATHERFORECAST table describe the value of each Sub-concept of the top-level class of Weather Attribute in the ontology. Sub-concept are Temperature, Wind, Wind direction, Humidity, Lightning, Precipitation, Solar Radiation, Atmospheric Pressure.

I. Code for creating the tables:

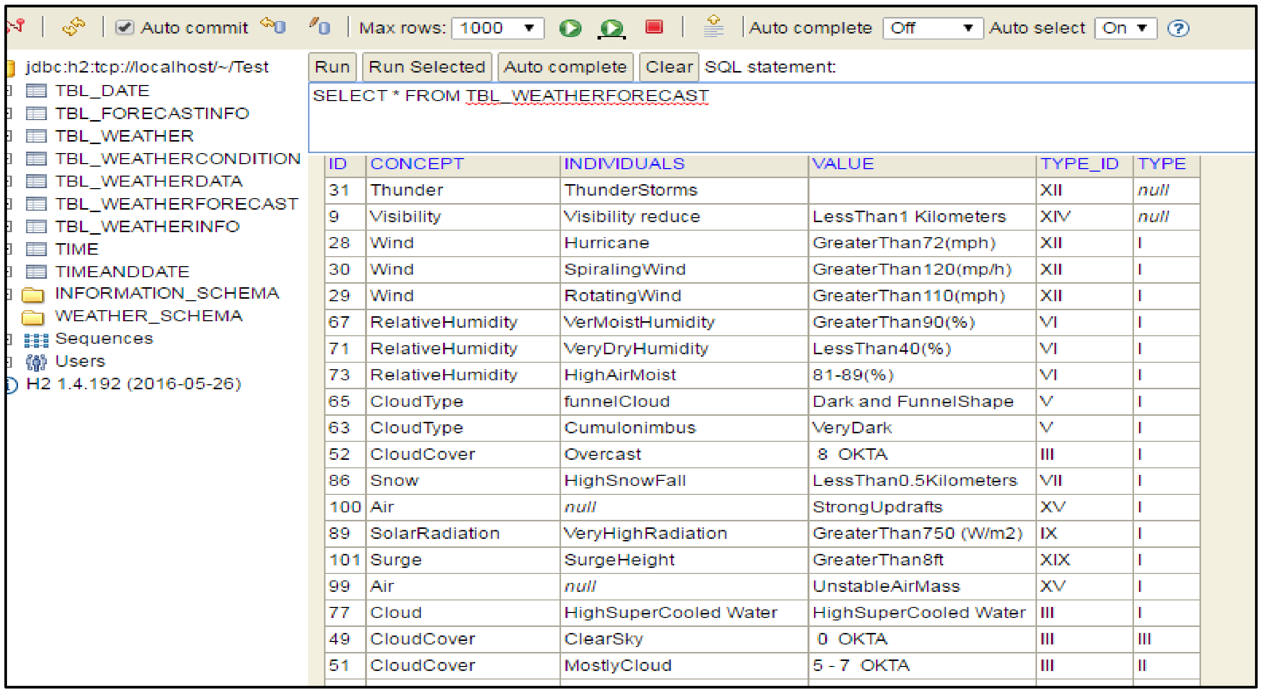
- **TBL_WeatherForecast**

```
CREATE TABLE tbl_WeatherForecast(
    ID int not null primary key,
    Concept varchar(40),
    Individuals varchar(40),
    Value varchar(40),
    Type_ID varchar(10)
);
```

Code for inserting the data into the tables

- **TBL_WeatherForecast**

```
INSERT INTO TBL_ATTRIBUTE
VALUES
(1,'Temperature','ExtremelyHotTemperature',' GreaterThan37(°C)', 'I'),
(2,'Temperature','HotTemperature',' 26-37(°C)', 'I'),
(3,'Temperature',' ColdTemperature',' 1-15(°C)', 'I'),
 --------------------------------
--------------------------------
(102,'Precipitation',' HeavyRainWithHail',' 21-50 (mm)', 'VII'),
```



```
SELECT * FROM TBL_WEATHERFORECAST
```

ID	CONCEPT	INDIVIDUALS	VALUE	TYPE_ID	TYPE
31	Thunder	ThunderStorms		XII	*null*
9	Visibility	Visibility reduce	LessThan1 Kilometers	XIV	*null*
28	Wind	Hurricane	GreaterThan72(mph)	XII	I
30	Wind	SpiralingWind	GreaterThan120(mp/h)	XII	I
29	Wind	RotatingWind	GreaterThan110(mph)	XII	I
67	RelativeHumidity	VerMoistHumidity	GreaterThan90(%)	VI	I
71	RelativeHumidity	VeryDryHumidity	LessThan40(%)	VI	I
73	RelativeHumidity	HighAirMoist	81-89(%)	VI	I
65	CloudType	funnelCloud	Dark and FunnelShape	V	I
63	CloudType	Cumulonimbus	VeryDark	V	I
52	CloudCover	Overcast	8 OKTA	III	I
86	Snow	HighSnowFall	LessThan0.5Kilometers	VII	I
100	Air	*null*	StrongUpdrafts	XV	I
89	SolarRadiation	VeryHighRadiation	GreaterThan750 (W/m2)	IX	I
101	Surge	SurgeHeight	GreaterThan8ft	XIX	I
99	Air	*null*	UnstableAirMass	XV	I
77	Cloud	HighSuperCooled Water	HighSuperCooled Water	III	I
49	CloudCover	ClearSky	0 OKTA	III	III
51	CloudCover	MostlyCloud	5 - 7 OKTA	III	II

Figure 5.17: Table for Weather Attribute.

In TBL_WeatherCondition table describe the Status for the concept of WeatherCondition.

II. Code for creating the tables:

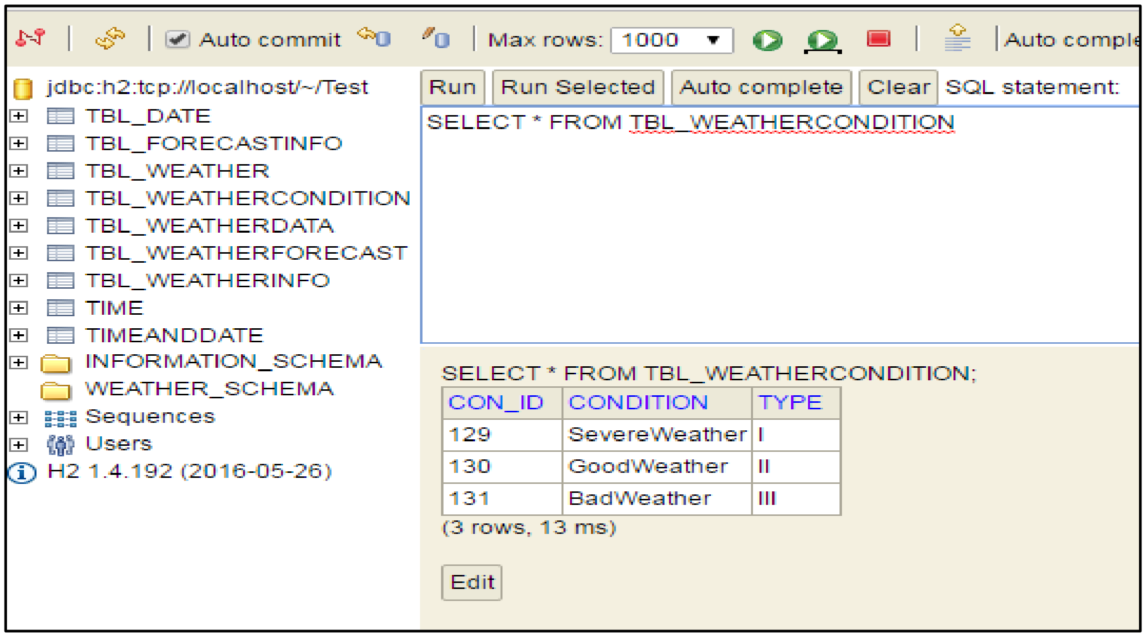
- **TBL_WeatherCondition**

```
CREATE TABLE TBL_WEATHERCONDITION(
   CON_ID int not null primary key,
   Condition varchar(40),
   Type varchar(10),
);
```

Code for inserting the data into the tables

- **TBL_WeatherCondition**

INSERT INTO **TBL_WeatherCondition**
VALUES
(empseq.nextval,‘ SevereWeather’, ‘I’),
(empseq.nextval,‘GoodWeather’, ‘II’),
(empseq.nextval,‘BadWeather’, ‘III’),



```
SELECT * FROM TBL_WEATHERCONDITION
```

SELECT * FROM TBL_WEATHERCONDITION;

CON_ID	CONDITION	TYPE
129	SevereWeather	I
130	GoodWeather	II
131	BadWeather	III

(3 rows, 13 ms)

Figure 5.18: Table for Weather Condition

In TimeAndDate table describe the Duration and Month for the concept of Time.

III. Code for creating the tables:

- **TBL_TIMEANDDATE**

  CREATE TABLE TBL_TIMEANDDATE(

  ID int not null primary key,

  Month varchar(40),

  hours int,);

- **TBL_TIMEANDDATE**

  INSERT INTO **TBL_TIMEANDDATE**

  VALUES

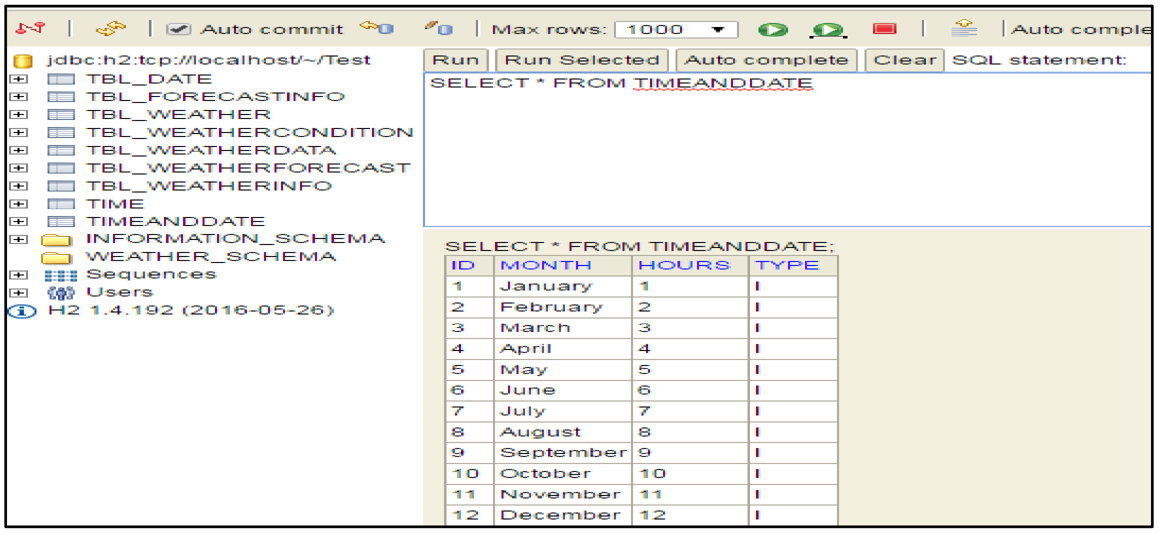
  (1, ‘January’, 1‘I’),

  (2, ‘February’, 2‘I’),

  (3, ‘March’, 3‘I’),

  (4, ‘April’, 5‘I’),

  (12, ‘December’, 42‘I’),



```
SELECT * FROM TIMEANDDATE
```

SELECT * FROM TIMEANDDATE;

ID	MONTH	HOURS	TYPE
1	January	1	I
2	February	2	I
3	March	3	I
4	April	4	I
5	May	5	I
6	June	6	I
7	July	7	I
8	August	8	I
9	September	9	I
10	October	10	I
11	November	11	I
12	December	12	I

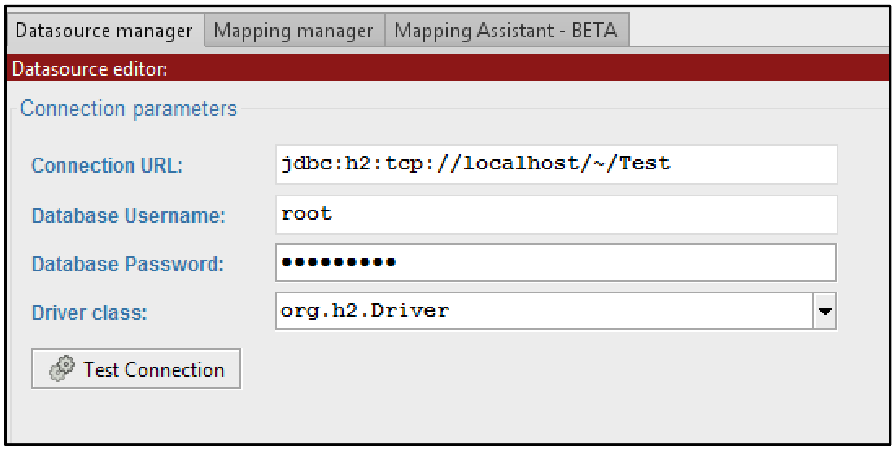
Figure 5.19: Table for Concept Time.

5.4 Ontop Mapping Tools

Ontop Mapping Tools are used to Map the data from H2 Relational Database to Ontology Hierarchy [4].

In Ontop Mapping tools there are three windows named Data Source Manager, Mapping Manager, Mapping Assistant-BETA.

The **Data source Manager** is the first sub-tab of the ontop Mappings tab. using this tab we will be able to manage data source declarations.



Connection URL:	jdbc:h2:tcp://localhost/~/Test
Database Username:	root
Database Password:	••••••••••
Driver class:	org.h2.Driver

Figure 5.20: Data Source Manager Window

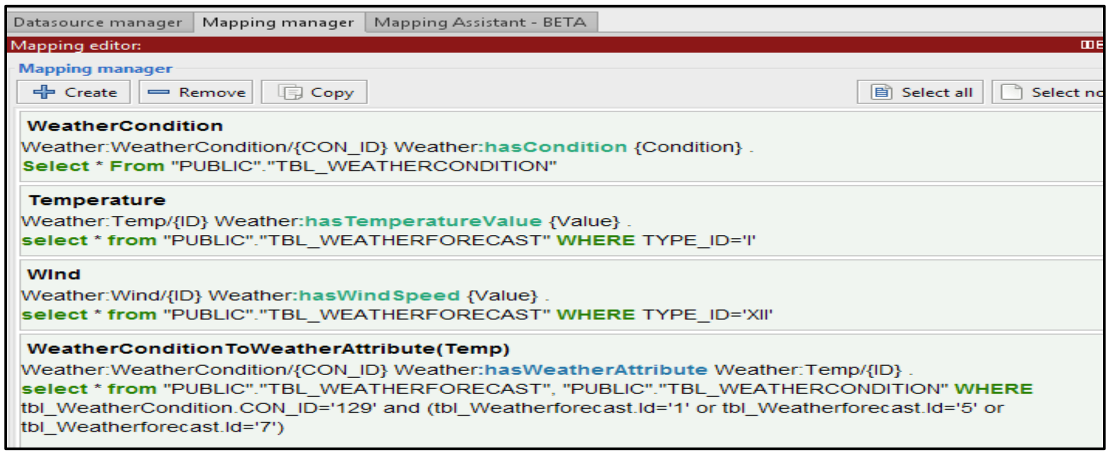
OBDA model will consist of one data source declaration. The source must be a JDBC data source and you must provide the usual JDBC connection parameters for them:

- The JDBC URL string.
- The database username,
- The database password, and
- The JDBC driver.

Once the information is filled you may use the "Test Connection" button to check that the system can establish a connection to the database. Make sure the connection is correct before querying using Sparql.

Mapping Manager

The Mapping Manager is where we will spent most time. Using the Mapping manager we can create and manage the mappings axioms for our data sources.



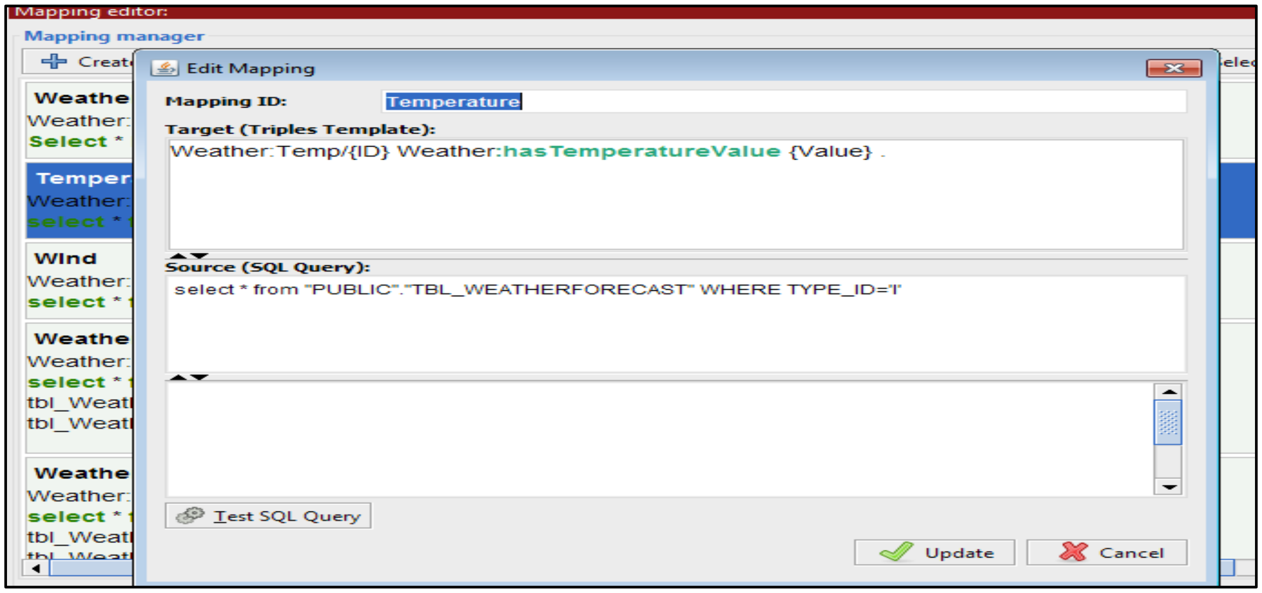
```
WeatherCondition
Weather:WeatherCondition/{CON_ID} Weather:hasCondition {Condition} .
Select * From "PUBLIC"."TBL_WEATHERCONDITION"

Temperature
Weather:Temp/{ID} Weather:hasTemperatureValue {Value} .
select * from "PUBLIC"."TBL_WEATHERFORECAST" WHERE TYPE_ID='I'

Wind
Weather:Wind/{ID} Weather:hasWindSpeed {Value} .
select * from "PUBLIC"."TBL_WEATHERFORECAST" WHERE TYPE_ID='XII'

WeatherConditionToWeatherAttribute(Temp)
Weather:WeatherCondition/{CON_ID} Weather:hasWeatherAttribute Weather:Temp/{ID} .
select * from "PUBLIC"."TBL_WEATHERFORECAST", "PUBLIC"."TBL_WEATHERCONDITION" WHERE
tbl_WeatherCondition.CON_ID='129' and (tbl_Weatherforecast.Id='1' or tbl_Weatherforecast.Id='5' or
tbl_Weatherforecast.Id='7')
```

Figure 5.21: Mapping Manager Window

Mapping axioms

An OBDA model must also contain one or more mapping axioms. A mapping axiom consists of three fields: mappingId, source and target. The mappingId is any string identifying the axiom, the source is an arbitrary SQL query over the database, and the target is a triple template that contains value that reference column names mentioned in the source query. For example, the following is a valid -ontop- mapping:

.



```
Mapping ID: Temperature
Target (Triples Template):
Weather:Temp/{ID} Weather:hasTemperatureValue {Value} .
Source (SQL Query):
select * from "PUBLIC"."TBL_WEATHERFORECAST" WHERE TYPE_ID='I'
```

Figure 5.22: Mapping Axioms in triple format

Here in the target, we have mapped the temperature value from the H2 database to the Sub-concept of temperature in our ontology in the triple formats.

Meaning of a mapping axiom

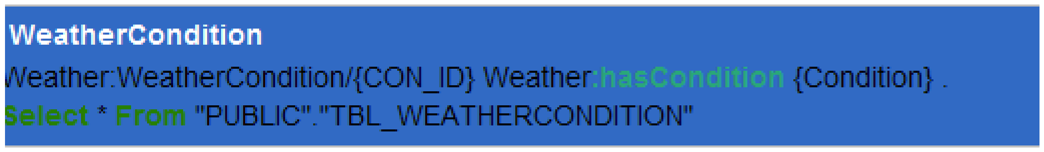
The purpose of the mapping axioms is to convert data in the specified data sources into a set of ABox assertions/RDF triples. From the figure 5.21 shows that we are converting Relation database tuples into RDF triples in following way.

Temp/{id} :hasTemperatureValue "49°*C*"

In this example "Temp/{id}" is a subject , ":hasTemperatureValue" is Predicates and "49°*C*" is a object which is a literal value. So we easily convert the Relational database tuples into a RDF triples.

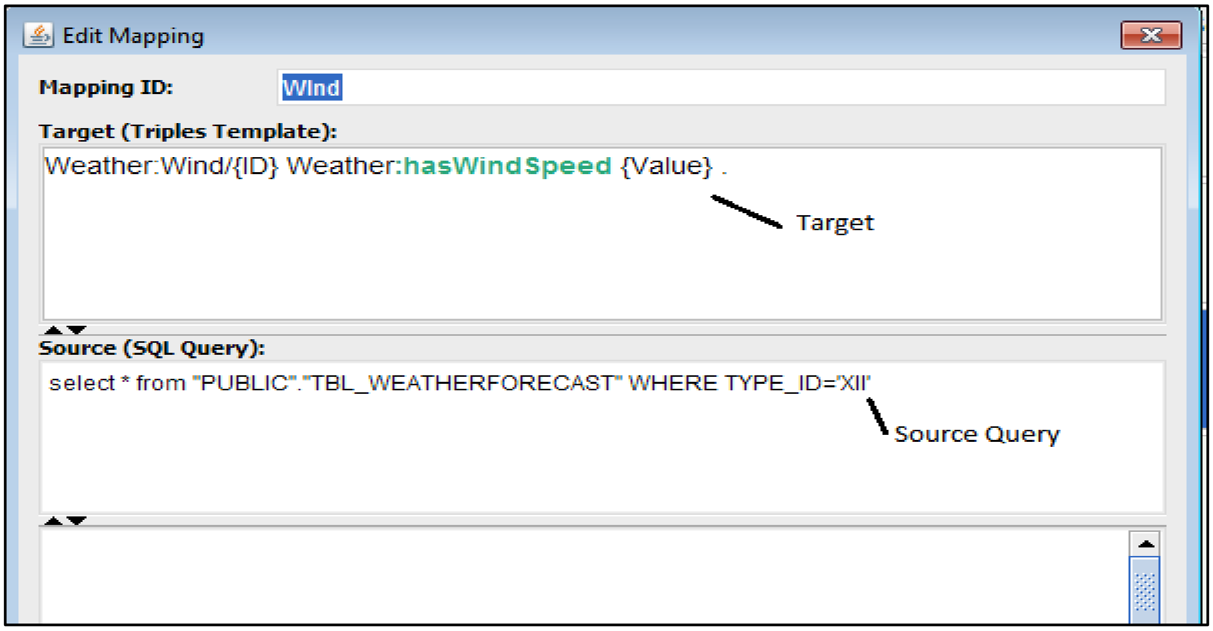
Creating a new mapping

All mappings are composed by a mappingID, a target and a source.



```
WeatherCondition
Weather:WeatherCondition/{CON_ID} Weather:hasCondition {Condition} .
Select * From "PUBLIC"."TBL_WEATHERCONDITION"
```

To create a new mapping for a data source simply click the + Create button.This will display the following dialog:



Mapping ID: Wind

Target (Triples Template):

```
Weather:Wind/{ID} Weather:hasWindSpeed {Value} .
```

Source (SQL Query):

```
select * from "PUBLIC"."TBL_WEATHERFORECAST" WHERE TYPE_ID='XII'
```

Figure 5.23: Mapping Editor

5.5 Sparql

After Mapping Completed Sparql is applied to the RDF graph to extract the Information from Data set according to the user defined query [3]. For Example

Maximum and minimum values

The following example shows a SPARQL query to obtain the maximum temperature value stored in the ontology; note that the query contains only the necessary triple patterns; any triple patterns that are not necessary to obtain the desired result are omitted, e.g. it is not necessary to ensure that ?p is an instance of weather:Temperature as the domain of the property weather:hasTemperature Value is defined to be weather:Temperature and therefore an ontop reasoner can infer that ?p must be an instance of weather:Temperature

```
SELECT (MAX(?t) AS ?t_max)
WHERE { ?p Weather:hasWeatherAttribute ?Temp
?Temp  weather:hasTemperatureValue ?v.
}
```

Figure 5.24: Sparql syntax to indentified Max temperature value.

Chapter 6

Conclusion

6.1 Summary

This project describes a weather data model based on OWL ontology. Ontology provides convenient vocabularies to a data source using the techniques of RDF and RDF Schema. RDF represent the data in the form of “subject” “predicate” and “object" and RDF schema define the class and object property with classified domain and range. As the overall goal of this project, weather phenomenon prediction represents a comprehensive ontological model to determine various weather conditions which are affected by different weather attributes. It is centered on a set of three top-level concepts, WeatherAttribue, WeatherCondition and time. WeatherAttribute has some sub-concepts such as temperature, humidity, precipitation and windspeed. WeatherCondition class is linked with each element of the WeatherAttribute class by the object property “hasWeatherAttribute”. And each element of the WeatherAttribute is linked with the WeatherCondition class by the object property “belongsToCondition” which is inverse of “hasWeatherAttribute”. Other top-level concept Time has two sub-concepts Interval and Instant which is linked with WeatherCondition by the object property “hasTimeandDate”. And each sub-concept has some individuals to represent the data value which is a granularity of the tree. This data value is mapped to our ontology from the relational database using the ontop mapping techniques. Ontop mapping tools support the JDBC drivers to access the ODBA. The key advantage of this mapping procedure is that no interpretation of data needs to be carried out to be stored as ontology instances. This has reduced a lot of work, because interpretation of data in existing data sources may cause some scalability issues with existing legacy applications. Data are fetching from the relational database in the triple formats. Sparql query is applied on the RDF triplestore to extract the information based on user defined.

Besides the work on weather phenomenon prediction itself, this project carries out extensive research regarding methodologies for developing ontologies. This methodology approaches help me to creating new ontologies from scratch are outlined, their characteristics are identified, and their suitability for applying them to the domain of weather phenomenon prediction to validate the ontology.

6.2 Limitation

There may be requirements that ontology is unable to fulfill due to certain limitations.

1. The Open World Assumption. For instance, OWL, which honors the Open World Assumption, cannot tell the absence of an instance of some concept. This leads to cases where an ontology fails to answer a competency question such as “Does this group only consist of women?”; just because the ontology does not contain an

individual which is a man, it does not mean that there is no man; the ontology can only tell that there is no man it knows about.
2. Ontology Building, maintenance, and reuse time consuming activities.
3. Communication between collaborators from different disciplines is difficult.
4. There is no Fixed method to create a ontology which is little bit confusing that whether building ontology giving the desired answer or not.
5. It is hardly avoidable to run into bugs that manifest themselves in the form of incomprehensible error messages.
6. Protégé has some issues with performance; the larger the ontology gets, the more time it takes to map the data from data source. This becomes a major issue for applications with huge ontology.
7. Response time of Sparql is higher than SQL. Maximum response time of sparql is 10 minutes to query the complex query.
8. In web ontology language it is not possible to define multi values for a functional property.
9. Reconstructing RDF graph non-trivial.

6.3 Future Work

1. Fuzzy logic, Neural network or Bayesian network technology has been introduce for numerical weather prediction.
2. More generalized development of the ontology to infer any kind of weather forecast.
3. We have not exercised the advantage of SPARQL at its best; further work must be done to handle more complex situations that may involve multiple ontology assertions.
4. In this project we are unable to work with SWRL to infer axioms. So in future we have more work on it to better infer axioms.
5. More research on how to determine preferences.
6. Develop a web application to visualizes the infer weather prediction based on user expectation.
7. Last but most important desired is to implement on Bangladesh Meteorology Department to predict more accurate weather forecast using this technology.

APPENDIX A

This appendix contains tables and listings that are referenced from other chapters.

A.1 Conceptualization tables for weather phenomenon prediction.

In order to keep the documentation of Weather Phenomenon clear, a set of tables is omitted from Section 5.2. This section contains these tables in case they are needed for reference

The tables in this section are:

- Concept dictionaries for Weather condition, WeatherAttribute, and Time (Table A.1); see Section 5.2.1 for details
- The Binary relations table in Table A.2; see Section 5.2.2 for details.
- The Instance attributes table in Table A.3; see Section 5.2.3 for details.
- The Class attributes table in Table A.4, see Section 5.2.4 for details.

Name	Instances	Relations
Time	Instant/1,Instant2,Instant/3,Instant/4,Instant/5,Instant/6, Instant/7,Instant/9,Instant/10,Instant/11,Instant/12,Instant/13,Instant/14,Instant/15,Interval/1,Interval/2,Interval/3,Interval/4,Interval/5,Interval/6,Interval/7,Interval/8,Interval/9,Interval/10,Interval/11,Interval/12,Interval/14, Interval/15,	hasTimeAndDate
Weather Attribute	Temp/1,Temp/2,Temp3,Temp4,Preci/78,preci/79,preci/80,preci/81,preci/82,Humi/70,,Humi/67,Humi/68,Humi/71,Radiation/89,Radiation/90,Radiation/91,CloudCover/49,CloudCover/50,CloudCover/51,CloudCover/61,Wind/16,Wind/17, Wind/18, Wind/19, Wind/20, Wind/21, Wind/22,Wind/23,Wind/25,Wind/27,Wind/30,direction	hasWeatherAttribute, belongsToCondition

	/32,direction/33,direction/35,direction/38,direction/40,direction/41,direction/43,direction/45,Ligthning/94,Lightning/96,Pressure/10,Pressure/11,Pressure/12,Pressure/14,Pressure/15,Surge/101,	
Weather Condition	WeatherCondition/129,WeatherCondition/130,WeatherCondition/131	hasWeatherAttribute, belongsToCondition, hasTimeAndDate

Table A.1: Concept dictionary for Weather condition, Weather Attribute, and Time.

Relation	Source Concept	Target Concept	Inverse Relation
hasWeatherAttribute	Weather Condition	Weather Attribute	belongsToCondition
belongsToCondition	Weather Attribute	Weather Condition	hasWeatherAttribute
hasTimeAndDate	WeatherCondition	Time	----

Table A.2: Binary relations table.

Data property name	Concept name	Value type	Value range	Unit
hasWindSpeed	Wind	xsd:decimal	[0, ∞)	mph
hasWDirection	WindDirection	xsd:string	[North,…………West)	---
hasTemperatureValue	Temperature	xsd:decimal	any values allowed	℃
hasPreciIntensity	Precipitation	xsd:decimal	[0, ∞)	mm/h
hasCloudCover	CloudCover	xsd:string	[Name)	------
hasHumiValue	Humidity	xsd:decimal	[0, 100)	Percentage
hasPressureValue	Atmospheric Pressure	xsd:decimal	[0, ∞)	hpa
hasRadiationValue	SolarRadiation	xsd:decimal	[0, ∞)	W/m2
hasDewValue	DewPoint	xsd:decimal	any values allowed allowed	℃
hasAirMass	AirMass	xsd:string	----	---------
hasLightningPresence	Lightning	xsd:boolean	[0,1)	-------
hasTunderOccure	Thunder	xsd:boolean	[0,1)	-------
hasPeriodOfDuration	Interval	xsd:dateTime	[0, ∞)	hours
hasPeriodOfDuration	Instant	xsd:string	(January,…….,December	months

Table A.3: Instance Data type table.

Super-Concept	Sub-Concept	Individual	Data property	Data property Value
Weather Attribute	Temperature	Temp/1	hasTemperatureValue	> 37
Weather Attribute	Temperature	Temp/2	hasTemperatureValue	26-37

Weather Attribute	Temperature	Temp/3	hasTemperatureValue	16-25
Weather Attribute	Temperature	Temp/4	hasTemperatureValue	< 0
Weather Attribute	Temperature	Temp/5	hasTemperatureValue	1-15
Weather Attribute	SolarRadiation	Radiation/89	hasRadiationValue	> 750
Weather Attribute	SolarRadiation	Radiation/90	hasRadiationValue	500 - 750
Weather Attribute	SolarRadiation	Radiation/91	hasRadiationValue	250 - 499
Weather Attribute	SolarRadiation	Radiation/92	hasRadiationValue	1 - 249
Weather Attribute	SolarRadiation	Radiation/93	hasRadiationValue	0
Weather Attribute	Precipitation	Preci/78	hasPreciIntensity	> 100
Weather Attribute	Precipitation	Preci/79	hasPreciIntensity	51-100
Weather Attribute	Precipitation	Preci/80	hasPreciIntensity	21-50
Weather Attribute	Precipitation	Preci/81	hasPreciIntensity	6-20
Weather Attribute	Precipitation	Preci/82	hasPreciIntensity	0
Weather Attribute	Precipitation	Preci/84	hasPreciIntensity	1-5
Weather Attribute	Wind	Wind/16	hasWindSpeed	0
Weather Attribute	Wind	Wind/17	hasWindSpeed	1-3
Weather Attribute	Wind	Wind/18	hasWindSpeed	4-7
Weather Attribute	Wind	Wind/19	hasWindSpeed	8-12
Weather Attribute	Wind	Wind/20	hasWindSpeed	13-18
Weather Attribute	Wind	Wind/21	hasWindSpeed	19-24
Weather Attribute	Wind	Wind/22	hasWindSpeed	25-31
Weather Attribute	Wind	Wind/23	hasWindSpeed	32-38
Weather Attribute	Wind	Wind/24	hasWindSpeed	39-46
Weather Attribute	Wind	Wind/25	hasWindSpeed	47-54
Weather Attribute	Wind	Wind/26	hasWindSpeed	55-63
Weather Attribute	Wind	Wind/27	hasWindSpeed	64-72
Weather Attribute	Wind	Wind/28	hasWindSpeed	> 72
Weather Attribute	Wind Direction	Direction/32	hasWdirection	North
Weather Attribute	Wind Direction	Direction/34	hasWdirection	North-NorthEast
Weather Attribute	Wind Direction	Direction/35	hasWdirection	NorthEast
Weather Attribute	Wind Direction	Direction/36	hasWdirection	East-NorthEast
Weather Attribute	Wind Direction	Direction/37	hasWdirection	East
Weather Attribute	Wind Direction	Direction/38	hasWdirection	East-SouthEast
Weather Attribute	Wind Direction	Direction/39	hasWdirection	SouthEast
Weather Attribute	Wind Direction	Direction/40	hasWdirection	South-SouthEast
Weather Attribute	Wind Direction	Direction/41	hasWdirection	South
Weather Attribute	Wind Direction	Direction/42	hasWdirection	South-SouthWest
Weather Attribute	Wind Direction	Direction/43	hasWdirection	SouthWest
Weather Attribute	Wind Direction	Direction/44	hasWdirection	West-SouthWest
Weather Attribute	Wind Direction	Direction/45	hasWdirection	West
Weather Attribute	Wind Direction	Direction/46	hasWdirection	West-NorthWest
Weather Attribute	Wind Direction	Direction/47	hasWdirection	NorthWest
Weather Attribute	Wind Direction	Direction/48	hasWdirection	North-NorthWest

Weather Attribute	Humidity	Humi/67	hasHumiValue	> 90
Weather Attribute	Humidity	Humi/68	hasHumiValue	71- 80
Weather Attribute	Humidity	Humi/69	hasHumiValue	50-70
Weather Attribute	Humidity	Humi/70	hasHumiValue	40-49
Weather Attribute	Humidity	Humi/71	hasHumiValue	< 40
Weather Attribute	Humidity	Humi/71	hasHumiValue	81-89
Weather Attribute	Lightning	Lightning/94	hasLightningPresenc	YES/NO
Weather Attribute	Thunder	Thunder/96	hasThunderOccuranc	YES/NO
Weather Attribute	CloudCover	Cloud/49	hasCloudCover	Clear Sky
Weather Attribute	CloudCover	Cloud/50	hasCloudCover	Partly Cloud
Weather Attribute	CloudCover	Cloud/51	hasCloudCover	Mostly Cloud
Weather Attribute	CloudCover	Cloud/52	hasCloudCover	OverCast
Weather Attribute	CloudCover	Cloud/53	hasCloudCover	UnKnownCloud
Weather Attribute	Atmospheric Pressure	Pressure/10	hasPressureValue	> 1028
Weather Attribute	Atmospheric Pressure	Pressure/11	hasPressureValue	1018-1028
Weather Attribute	Atmospheric Pressure	Pressure/12	hasPressureValue	< 998
Weather Attribute	Atmospheric Pressure	Pressure/13	hasPressureValue	1008-1017
Weather Attribute	Atmospheric Pressure	Pressure/14	hasPressureValue	998- 1007
Time	Instant	Instant/1	hasPeriodOfMonth	January
Time	Instant	Instant/2	hasPeriodOfMonth	February
Time	Instant	Instant/3	hasPeriodOfMonth	March
Time	Instant	Instant/4	hasPeriodOfMonth	April
Time	Instant	Instant/5	hasPeriodOfMonth	May
Time	Instant	Instant/6	hasPeriodOfMonth	June
Time	Instant	Instant/7	hasPeriodOfMonth	July
Time	Instant	Instant/8	hasPeriodOfMonth	August
Time	Instant	Instant/9	hasPeriodOfMonth	September
Time	Instant	Instant/10	hasPeriodOfMonth	October
Time	Instant	Instant/11	hasPeriodOfMonth	November
Time	Instant	Instant/12	hasPeriodOfMonth	December
Time	Interval	Interval/1	hasPeriodOfDuration	1
Time	Interval	Interval/2	hasPeriodOfDuration	3
Time	Interval	Interval/3	hasPeriodOfDuration	6
Time	Interval	Interval/4	hasPeriodOfDuration	9

Time	Interval	Interval/5	hasPeriodOfDuration	12
Time	Interval	Interval/6	hasPeriodOfDuration	15
Time	Interval	Interval/7	hasPeriodOfDuration	18
Time	Interval	Interval/8	hasPeriodOfDuration	21
Time	Interval	Interval/9	hasPeriodOfDuration	24
Time	Interval	Interval/10	hasPeriodOfDuration	36
Time	Interval	Interval/11	hasPeriodOfDuration	42
Time	Interval	Interval/12	hasPeriodOfDuration	76
Weather Condition	------	Condition/1	hasCondition	Severe Condition
Weather Condition	---------	Condition/2	hasCondition	Good Condition
Weather Condition	-----------	Condition/3	hasCondition	Bad Condition

Table A.4: Class attributes table

A.2 Figure

Sparql syntax and output result

Syntax

```
PREFIX : <http://localhost/Weather.owl#>

SELECT DISTINCT ?Condition  ?Factors  ?ThunderStorm

WHERE{

{?WeatherCondition :hasCondition ?Condition. ?WeatherCondition :hasWeatherAttribute ?Factors.?Factors
:hasHighWindSpeed "64-72(mph)". ?Factors :hasHighWindSpeed ?ThunderStorm. }

UNION{?WeatherCondition :hasCondition ?Condition. ?WeatherCondition :hasWeatherAttribute ?Factors.
?Factors :hasHighPreciIntensity "21-50 (mm)". ?Factors :hasHighPreciIntensity ?ThunderStorm.}

UNION{?WeatherCondition :hasCondition ?Condition. ?WeatherCondition :hasWeatherAttribute ?Factors.

?Factors :hasFuriousCloudType "Cumulonimbus".?Factors :hasFuriousCloudType ?ThunderStorm.}

UNION{?WeatherCondition :hasCondition ?Condition. ?WeatherCondition :hasWeatherAttribute ?Factors.

?Factors :hasLightningPresence ?ThunderStorm.}

UNION{?WeatherCondition :hasCondition ?Condition.?WeatherCondition :hasWeatherAttribute ?Factors.

?Factors :hasTunderOccur ?ThunderStorm.}}
```

```
PREFIX : <http://localhost/Weather.owl#>
SELECT DISTINCT  ?Description ?Factors ?VeryHumidWeather
WHERE{
{ ?WeatherCondition :hasCondition ?Description.?WeatherCondition :hasWeatherAttribute ?Factors.
?Factors :hasAverageTemp ?VeryHumidWeather.
 FILTER regex(str(?VeryHumidWeather ), "26-37")
}UNION{?WeatherCondition :hasCondition ?Description. ?WeatherCondition :hasWeatherAttribute ?Factors.
?Factors :hasHighHumi ?VeryHumidWeather.
FILTER regex(str(?VeryHumidWeather ), "81-89") }
UNION{?WeatherCondition :hasCondition ?Condition. ?WeatherCondition :hasWeatherAttribute ?Factors.
```

Figure 2: Sparql Syntax for prediction of factors for VeryHumidWeather

```
PREFIX : <http://localhost/Weather.owl#>
SELECT DISTINCT  ?Condition ?Factors  ?DrizzleRain ?DropletSize
WHERE{
{
?WeatherCondition :hasCondition ?Condition. ?WeatherCondition :hasWeatherAttribute ?Factors.
?Factors :hasNormalPreciIntensity ?DrizzleRain. FILTER regex(str(?DrizzleRain), "1 - 5").}
UNION{?WeatherCondition :hasCondition ?Condition. ?WeatherCondition :hasWeatherAttribute ?Factors.
?Factors :hasDropLetSize ?DropletSize.}
```

Figure 2: Sparql Syntax for prediction of factors for Drizzle

```
PREFIX : <http://localhost/Weather.owl#>
PREFIX Weather: <http://localhost/Weather.owl#>
SELECT DISTINCT ?Description ?Factors ?TropicalCyclone
WHERE
{
{   ?WeatherCondition   :hasCondition   ?Description.?WeatherCondition   :hasWeatherAttribute   ?Factors.
?Factors :hasLowPressureValue "998 - 1007(hPa)". ?Factors :hasLowPressureValue ?Value.
bind(concat( "LowPressureAtCentralPoint, ", " , " , ?Value) as ?TropicalCyclone  )}

UNION{ ?WeatherCondition :hasCondition ?Description. ?WeatherCondition :hasWeatherAttribute ?Factors.
?Factors :hasHighHumi ?TropicalCyclone. FILTER regex (str(?TropicalCyclone), "81-89")}

UNION{ ?WeatherCondition :hasCondition ?Description. ?WeatherCondition :hasWeatherAttribute ?Factors.
?Factors  :hasHighWindSpeed  "GreaterThan72(mph)".  ?Factors   :hasHighWindSpeed  ?Value.  bind(concat(
"SpiralingWind", ", " , ?Value) as ?TropicalCyclone  )}

UNION{ ?WeatherCondition :hasCondition ?Description. ?WeatherCondition :hasWeatherAttribute ?Factors.
?Factors    :hasAbnormalTemp    "GreaterThan26(°C)".    ?Factors    :hasAbnormalTemp    ?Valuebind(concat(
"SeaSurfaceTempreature", ", " , ?Value) as ?TropicalCyclone  ) }

UNION{ ?WeatherCondition :hasCondition ?Description. ?WeatherCondition :hasWeatherAttribute ?Factors.
?Factors :hasHighPreciIntensity "GreaterThan100(mm/h)".?Factors :hasHighPreciIntensity  ?TropicalCyclone.
}

UNION{ ?WeatherCondition :hasCondition ?Description. ?WeatherCondition :hasWeatherAttribute ?Factors.
?Factors :hasSurgeHeight ?TropicalCyclone  }

UNION

{?WeatherCondition :hasCondition ?Description.?WeatherCondition :hasTimeAndDate ?X.

?WeatherCondition :hasTimeAndDate ?Y.  ?X :hasPeriodOfMonth "April" . ?X :hasPeriodOfMonth ?Month1.
?Y  :hasPeriodOfMonth  "May"  .  ?Y  :hasPeriodOfMonth  ?Month2.  BIND  (concat  (?X,  "  ,  "  ,  ?Y)  as
?Factors)BIND (concat ( "Early Summar ",  ?Month1, " , " , ?Month2) as ?TropicalCyclone)}

UNION{?WeatherCondition    :hasCondition    ?Description.    ?WeatherCondition    :hasTimeAndDate    ?A.
?WeatherCondition   :hasTimeAndDate   ?B.   ?A   :hasPeriodOfMonth   "October"   .   ?A   :hasPeriodOfMonth
?Month3. ?B :hasPeriodOfMonth "November" . ?B :hasPeriodOfMonth ?Month4.BIND (concat (?A, " , " , ?B)
as ?Factors)BIND (concat ( "Late RainySeason ",  ?Month3, " , " , ?Month4) as ?TropicalCyclone)}}

}
```

Figure 4: Sparql Syntax for prediction of factors for Cyclone.

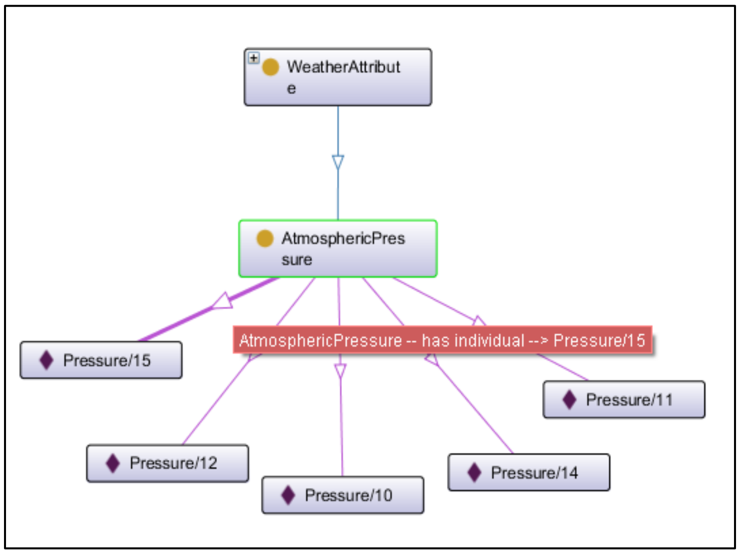


Figure 5: Individual of Atmospheric Pressure

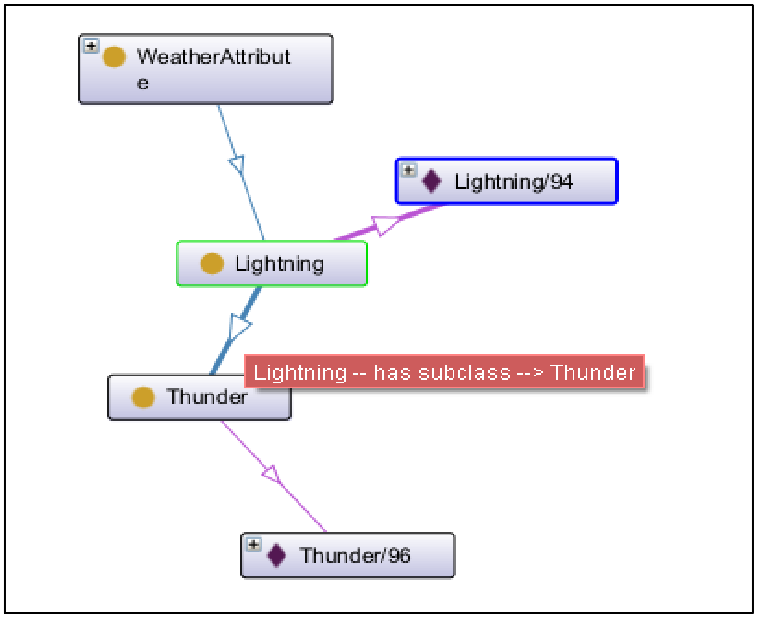
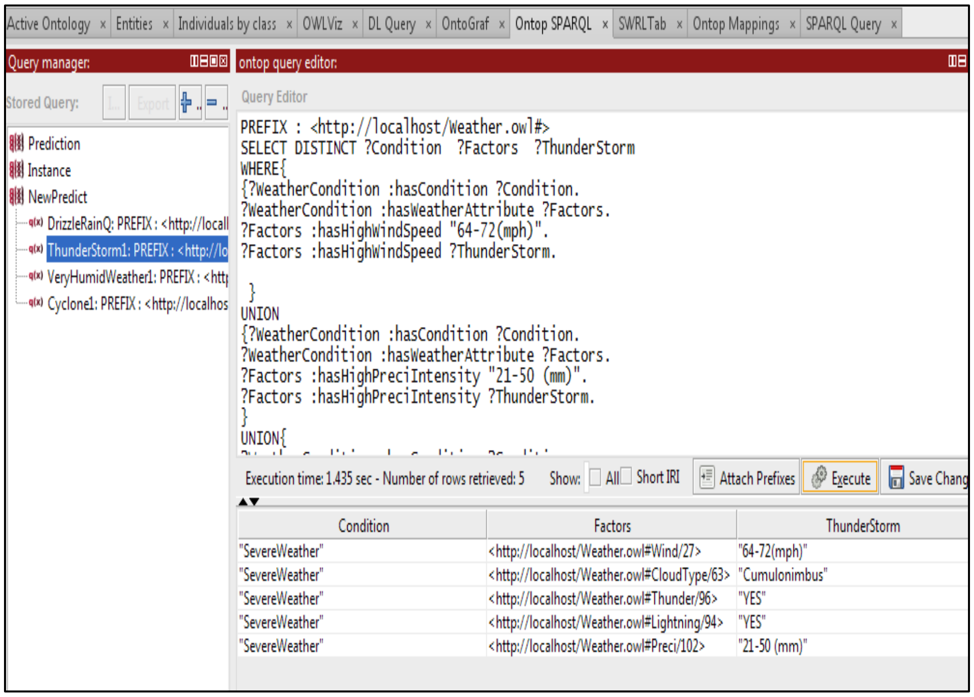


Figure 6: Individual of Lightning

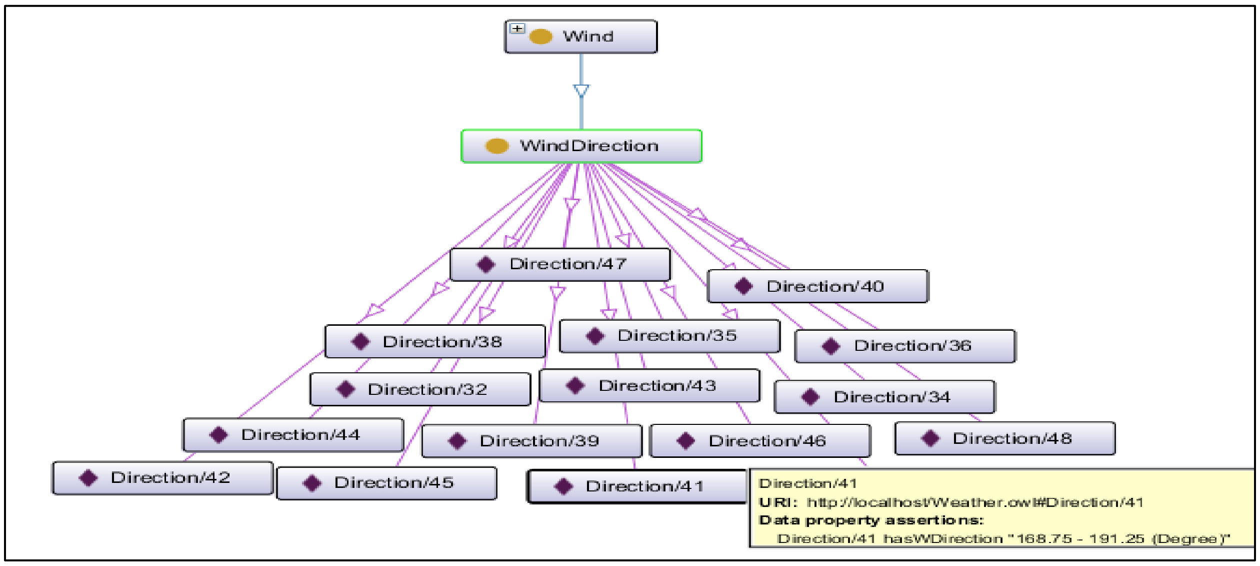


```
PREFIX : <http://localhost/Weather.owl#>
SELECT DISTINCT ?Condition  ?Factors  ?ThunderStorm
WHERE{
{?WeatherCondition :hasCondition ?Condition.
?WeatherCondition :hasWeatherAttribute ?Factors.
?Factors :hasHighWindSpeed "64-72(mph)".
?Factors :hasHighWindSpeed ?ThunderStorm.

 }
UNION
{?WeatherCondition :hasCondition ?Condition.
?WeatherCondition :hasWeatherAttribute ?Factors.
?Factors :hasHighPreciIntensity "21-50 (mm)".
?Factors :hasHighPreciIntensity ?ThunderStorm.
}
UNION{
```

Execution time: 1.435 sec - Number of rows retrieved: 5

Condition	Factors	ThunderStorm
"SevereWeather"	<http://localhost/Weather.owl#Wind/27>	"64-72(mph)"
"SevereWeather"	<http://localhost/Weather.owl#CloudType/63>	"Cumulonimbus"
"SevereWeather"	<http://localhost/Weather.owl#Thunder/96>	"YES"
"SevereWeather"	<http://localhost/Weather.owl#Lightning/94>	"YES"
"SevereWeather"	<http://localhost/Weather.owl#Preci/102>	"21-50 (mm)"

Figure 7: ontop Sparql Window



Direction/41
URI: http://localhost/Weather.owl#Direction/41
Data property assertions:
Direction/41 hasWDirection "168.75 - 191.25 (Degree)"

Figure 8: Individuals of Wind Direction

References:

1. Paul Staroch Developed A Weather Ontology for Predictive Control in Smart Homes to the Faculty of Informatics at the Vienna University of Technology, Software Engineering & Internet Computing, 29.08.2013
2. Natalya F. Noy and Deborah L. McGuinness. Ontology Development 101: A Guide Creating Your First Ontology. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March 1. 2001
3. .Hai Dong, RMIT University, Farookh Khadir Hussain, Application of Protégé and SPARQL in the Field of Project Knowledge Management"University of Technology, Sydney, Elizabeth Chang, Curtin University, Conference Paper, January 2007, "
4. Timea Bagosi, Diego Calvanese, Sarah Komla-Ebri, Davide Lanti, Martin Rezk, Mindaugas Slusnys, Guohui Xiao, "The *Ontop* Framework for Ontology Based Data Access"Free University of Bozen-Bolzano, Bolzano, Italy, Joseph Hardi, Obidea Technology, Jakarta, Indonesia, Mariano, Rodriguez-Muro, IBM T.J. Watson Research Center, NY, USA 18 November 2014.
5. Mariano Rodr__guez-Muro1, Roman Kontchakov2 and Michael Zakharyaschev2 Free University of Bozen-Bolzano, Italy Department of Computer Science and Information Systems, Birkbeck, University of London
6. Sumi Helal, William Mann, Hicham El-Zabadani, Jeffrey King, Youssef Kaddoura, and Erwin Jansen. The Gator Tech Smart House: A Programmable Pervasive Space. Computer,38(3):50–60, March 2005.
7. Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti Martin Rezk, Mariano Rodriguez-Muro , and Guohui Xiao Ontop: Answering SPARQL Queries over Relational Databases, Free University of Bozen-Bolzano.13 July 2002
8. John Bally Tal Boneh**Ann E. Nicholson***Kevin B. Korb; Developing An Ontology for the Meteorological Forecasting Process, School of Computer Science and Software Engineering Monash University, 3800, Victoria, Australia
9. Ramar Kaladevi 1, Gurunathan Geetha 2, P. Narayanasamy Developing An Ontology for the Meteorological Forecasting Process, Research scholar, Computer science and engineering department, Jerusalem College ofEngineering, Chennai, Tamilnadu, India, Vol. 39, Nº 1, 185 - 192, 2016
10. Rupal Gupta, Ruchika Gupta, Ashish Bishnoi ROLES AND USAGES OF RDF, SPARQL, ONTOLOGY TOOLS IN SEMANTIC WEB, *CCSIT, Teerthanker Mahaveer University, Moradabad Bansal Institute, Rohini,* 28 July 2016.
11. Blake Middleton, James Halbert, and Frank P. Coyle, Security Impacts on Semantic Technologies in the Coming Decade Southern Methodist University, Dallas TX 75205, USA, 5-sept 2003.

12. Ghislain Atemezing, Oscar Corcho b, Daniel Garijo b, José Mora b, María Poveda- Villalón b, Pablo Rozas c, Daniel Vila-Suero b and Boris Villazón-Terrazas Transforming Meteorological Data into Linked Data, EURECOM, Multimedia Department, Campus SophiaTech, France, 1-5-2011

13. http://www.sciencedirect.com/science/article/pii/S1674237015302520 [Access date 10-12-2016]
14. http://www.h2database.com/h2.pdf [Access date 18-02-2017]
15. http://ldc.usb.ve/~yudith/docencia/UCV/ScientificAmerican_FeatureArticle_TheSemanticWeb_May2001.pdf [Access date 25-02-2017]
16. https://github.com/ontop/ontop/wiki/ontopProMappingsTab [Access date 16-03-02-2017]
17. https://en.wikipedia.org/wiki/Weatherforecasting [Access date 18-11-2016]
18. https://www.w3.org/TR/sparql11-query/ [Access date 17-12-2016]
19. https://dior.ics.muni.cz/~makub/owl/ [Access date 1-02-2017]
20. https://en.wikipedia.org/wiki/Cyclone [Access date 15-02-2017]
21. https://en.wikipedia.org/wiki/Thunderstorm [Access date 10-01-2017]
22. https://en.wikipedia.org/wiki/Drizzle [Access date 15-12-2016]