Spring 5-11-2012

Spatial Ontology for the Production Domain of Petroleum Geology

Dickson M. Liadey
GEORGIA STATE UNIVERSITY

SPATIAL ONTOLOGY FOR THE PRODUCTION DOMAIN OF PETROLEUM GEOLOGY

by

DICKSON M. LIADEY

Under the Direction of Hassan A. Babaie

ABSTRACT

The availability of useful information for research strongly depends on well structured relationships between consistently defined concepts (terms) in that domain. This can be achieved through ontologies. Ontologies are models of the knowledge of specific domain such as petroleum geology, in a computer understandable format. Knowledge is a collection of facts. Facts are represented by RDF triples (subject-predicate-object). A domain ontology is therefore a collection of many RDF triples, which represent facts of that domain. The SWEET ontologies are upper or top-level ontologies (foundation ontologies) consisting of thousands of very general concepts. These concepts are obtained from of Earth System science and include other related concepts. The work in this thesis deals with scientific knowledge representation in which the SWEET ontologies are extended to include wider, more specific and specialized concepts used in Petroleum Geology. Thus Petroleum Geology knowledge modeling is presented in this thesis.

INDEX WORDS: Ontology, Semantic Web, OWL, Class, Subclass, Petroleum Geology, SWEET ontology

SPATIAL ONTOLOGY FOR THE PRODUCTION DOMAIN OF PETROLEUM GEOLOGY

by

DICKSON M. LIADEY

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in the College of Arts and Sciences

Georgia State University

2012

SPATIAL ONTOLOGY FOR THE PRODUCTION DOMAIN OF PETROLEUM GEOLOGY

by

DICKSON M. LIADEY

Committee Chair: Hassan A. Babaie

Committee: Crawford Elliott

Daniel Deocampo

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

May 2012

DEDICATION

I dedicate this thesis to Jesus Christ my Lord and Righteous Savior: The WORD OF GOD and One through whom all things were created (JOHN 1:1-5; ACTS 3:15). I am indeed thankful to Christ my Lord for the Blessed gift of life, for His provision and protection.

I also dedicate this thesis to my lovely wife, Adeline Liadey for her unflinching support, love and encouragement. Finally, I dedicate this thesis to our sweet and lovely daughter Christelle Liadey, for the warmth and joy she brings.

ACKNOWLEDGEMENTS

My profound gratitude goes to Dr. Hassan A. Babaie, my thesis advisor and the chair of my thesis committee for supervising this thesis. I deeply appreciate his weighty contribution to the success of this piece of work. In the Fall of 2010, he introduced and taught a course in Geoinformatics in the Department of Geosciences. The course content involved languages such as XML, RDF, RDF(S) and OWL, and the general method of building ontologies. The lessons from this class gave a lot of impetus to this thesis and I am indeed deeply grateful for Dr. Babaie's supports in making this work a great success.

I am also extremely thankful for the valuable contributions of Dr. W. Crawford Elliot, and Dr. Daniel M. Deocampo, the other members on my thesis committee. Special thanks again to Dr. Daniel M. Deocampo who directed my studies in Petroleum Geology.

My sincere gratitude also goes to Dr. Timothy La Tour and to the entire faculty of the Geoscience Department of Georgia State University, whose impartation of knowledge has greatly contributed to my academic success.

Finally, grateful thanks to my fellow students and staff of the Department of Geoscience, for the diverse support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This thesis presents a cogent re-engineering of an aspect of the existing upper-level Semantic Web for Earth and Environmental Terminology (SWEET) ontologies (http://sweet.jpl.nasa.gov/2.3) to suit its efficient usage by domain experts in the field of Petroleum Geology. An ontology is a representation of our knowledge of a domain in a machine (computer) understandable format (Yu, 2010). It also consistently defines the terms used to describe and represent the domain or area of knowledge (Heflin, 2009). The knowledge of a domain is the statements of facts of that domain. For example, in Petroleum Geology, statements of facts include: petroleum is composed of hydrocarbons and other elements; a petroleum trap generally consists of a source rock, reservoir rock, and cap rock; oil is a kind of petroleum; natural gas is a kind of petroleum; oil is the same as crude oil or simply crude. This knowledge (statements of facts) can be encoded through established relationships (superclass-subclass relationships) between concepts (classes), the properties describing these concepts, and the use of certain reserved words **(**e.g., owl:class) in the form of a subject-predicate-object format, called RDF triple.

Gruber (1993) defines ontology as an explicit formal specification of conceptualization. Babaie (2011), explains that in the Gruber's definition of ontology, "explicit" means that each and every fact is explicitly (openly, clearly, plainly, unambiguously) translated into RDF triples or class hierarchies (e.g., rdfs:subClassOf). "Formal" means that ontology is machine (computer) readable because ontology is written in standard machine readable languages of RDF, RDFS, and OWL. "Specification" means that you make statements of the relations with

the language constructs, for example when you say that SedimentaryRock is a Rock; you are explicitly specifying that SedimentaryRock (a concept) is a Rock (another concept). The is-A is the specific relation (specification). "Conceptualization" in this context means that the relations between classes (concepts) are represented with OWL constructs.

The concepts in the ontology are structured in a hierarchical superclass-subclass relationship (is-A relationship). The existing SWEET ontologies upon which the extensions are made, are developed by NASA's Jet Propulsion Lab for Earth System science. These publicly available SWEET ontologies are written in OWL (Web Ontology Language) and include several thousand terms, spanning the broad extent of Earth System science and related concepts (Raskin and Pan, 2005).

In the model, presented in this thesis, top-level concepts from the SWEET ontologies are used and specialized (extended) for the first time to build an ontology for the Petroleum Geology domain. The re-engineering of the SWEET ontologies is to provide an in-depth resource base of relevant concepts in Petroleum Geology and thus enhance a more efficient usage among experts in the Petroleum Geology domain. This ontology is about spatial objects, it does not deal with processes.

1.1 General method of constructing a scientific ontology

Noy and McGuiness (2003) proposed a set of ordered steps as a general method in building or constructing ontologies, which are given in a modified form in the following paragraphs.

The first step in the ontology building process is the determination of the domain and scope of the ontology. The domain is the field of study or discipline under consideration in the ontology, which in this thesis is the Petroleum Geology. The domain is usually very large, so one would want to define a scope of the ontology, which is the specific part of the domain knowledge of interest to capture and model (for example an aspect petroleum production). The scope of the ontology is often determined through the use of competency questions (Gruninger and Fox, 1995) which are the list of questions which a knowledge represented by the ontology should be able to answer. The scope of this ontology is on the aspect of Petroleum Geology which concern petroleum production (Petroleum Geology mainly involves petroleum exploration and petroleum production). This stage strongly depends on the purpose of the ontology.

The second step is the acquisition of domain knowledge. This knowledge attainment involves the extraction of relevant terms or concepts used in the domain of discourse. These terms (concepts) are nouns and verbs used as vocabularies in the domain.

The third step is to consider the reuse of existing ontologies in the scoped domain of interest. Some ontologies may be located at: www.ksl.stanford.edu/soft-ware/ontoligua and the DAML ontology library at: www.daml.org/ontologies (Loh et al., 2008). In that case, the available ontologies must be reused as much as possible.

The fourth step involves the identification of object (spatial) or processes classes, superclasses, and subclasses (i.e., the identification of hierarchical structure among object and process entities). Superclasses represent higher level concepts and subclasses are specializations of these concepts. For example, in the Petroleum Geology domain, one could identify PetroleumTrap as a class. This class (PetroleumTrap) could have StratigraphicTrap and

StructuralTrap as subclasses (specializations). To further extend this, a StructuralTrap for instance would have AnticlinalTrap and FaultTrap as subclasses (types). The approach adopted in this thesis for the hierarchy of classes is the *bottom-up method*, which starts with defining more specific, specialized concepts which are then combined or grouped to form more general concepts higher in the hierarchy (Uschold and Gruninger, 1996).

The fifth step involves identifying properties which take non-object values (datatype properties), which relate an instance of a class to the predefined XML data types or RDF literals, such as string, date, integer etc. Porosity, permeability, and thickness of a reservoir rock are all datatype properties. In addition to these datatype properties, object properties (e.g., containedIn, locatedIn, composedOf etc.) are identified. Object properties relate instances of different classes to each other. For example, the containedIn property relates instances of Oil and PetroleumReservoir classes. It is important at this stage to determine the characteristics of the properties, i.e., whether they are functional, inverse functional, symmetric, reflexive, transitive, etc. Also the cardinality (number of objects participating in a property) and value restrictions for the properties may be identified (Babaie, 2011). Property characteristics and restrictions are discussed in detail in Chapter 5.

The sixth and final step is to construct the scoped domain ontology using the identified concepts and relationships. The ontology may become populated with instances (individuals) of classes and their attributes with values.

CHAPTER 2

THE PETROLEUM GEOLOGY DOMAIN

2.1 Oil and Natural Gas formation

Petroleum Geology is a branch of Geology which deals with the application of Geology to the exploration for and production of oil and gas (Selley, 1998). Petroleum mainly forms from buried organic matter in a steadily subsiding sedimentary basin (Selley, 1998). The major source of organic matter from which petroleum forms is known as sapropel. Sapropel is predominantly fine plant and animal organic matter from marine microorganisms: phytoplankton (microscopic plants), zooplankton, and pores which accumulate in oxygen-deficient environments such as lakes, lagoons, or marine basins (Boggs, 2006). Such oxygen-poor environments could come about as a result of poor water circulation to replenish any oxygen used for instance in a decay process or if so much organic matter is supplied that, it completely overwhelms any available oxidants (Boggs, 2006). As the organic matter begins to decay by microbial activity any available oxygen is used in the decay process. The remaining organic matter is therefore preserved and buried under newer layers of marine sediments (Murck et al., 1996).

Tissot (1977) defines three major phases surface organic matter under goes during burial resulting in the formation of oil and gas. These phases are diagenesis, catagenesis and metagenesis. Diagenesis involves biogenic decay aided by bacteria, and reactions that are not biogenic in nature. During this process, methane, carbon dioxide, and water are released resulting in the formation of kerogen and a reduction in oxygen content. With increasing burial, temperature and pressure increases resulting first in the formation of oil from the kerogen and

later gas (natural gas). Kerogen therefore matures into oil and natural gas. Oil is generated at 60 and 120 ^{0}C, and thermogenic gas is generated between 120 and 225 ^{0}C. The hydrogen:carbon ratio of the kerogen declines in the process of generating oil and gas. This phase is known as catagenesis. With still further burial resulting in increasing higher temperatures and pressures methane is largely expelled and the hydrogen:carbon ratio further declines eventually resulting the formation of carbon in the form of graphite. This phase is known as metagenesis.

Petroleum encompasses naturally occurring liquid and gaseous hydrocarbons (oil and natural gas), and also semisolid hydrocarbons, commonly called tar (Murck et al., 1996). Oil and natural gas are the two main forms of Petroleum (Murck et al., 1996). Selley (1998) mentions that petroleum exploration is largely concerned with the search for oil and natural gas.

According to selley (1998), The American Petroleum Institute (API), the American Association of Petroleum Geologist (AAPG) and the Society of Petroleum Engineers (SPE) jointly define crude oil "as a mixture of hydrocarbons that existed in the liquid phase in natural underground reservoirs and remains liquid at atmospheric pressure after passing through surface separating facilities". Liquid hydrocarbons are referred to as oil, crude oil, or simply crude (Selley, 1998). Crude oils vary in appearance, sulfur content, surface viscosities, refractive indices, density and specific gravity. These variations are sometimes due to temperature variations and to the various hydrocarbons present in crude oil (Selley, 1998). The density of oil is often expressed in the API gravity units (^{0}API) defined by the American Petroleum Institute (API). By this definition, light oils have API gravities of greater than 40^0 and heavy oils have API gravities of less than 10^0 (Selley, 1998).

Also according to to Selley (1998), in the oil industry natural gas is defined as "a mixture of hydrocarbons and varying quantities of nonhydrocarbons that exists either in the gaseous phase or in solution with crude oil in natural underground reservoirs". This is the adopted definition by the American Petroleum Institute (API), the American Association of Petroleum Geologist (AAPG) and the Society of Petroleum Engineers (SPE). The above authorities again subclassify natural gas into dissolved, associated, and nonassociated gas. Dissolved gas is natural gas that exists in solution in crude oil in the reservoir. Associated gas, also known as gas cap gas is natural gas that overlies and is in contact with crude oil in the reservoir. Nonassociated gas is natural gas that is not in contact with significant quantities of crude oil in the reservoir (Selley, 1998). Such reservoirs containing nonassociated gas may contain some crude oil but not in significant quantities (Hyne, 2001). Natural gas is also described as sweet or sour. Sweet natural gas (sweet gas) is natural gas containing very little or no sulphur or sulphur compounds (e.g., hydrogen sulfide). Sour natural gas (sour gas) is natural gas containing significant quantities of sulphur or sulphur compounds (Hyne, 2001). Selley (1998) mentions that gases are described as sweet or sour based on the absence or presence respectively of hydrogen sulfide. Selley (1998) also mentioned that gases may be classified as dry gas if they contain less than 0.10 gal/ $1000ft^3$ of condensate and as wet gas if they contain more than 0.30 gal/ $1000ft^3$ of condensate, and that chemically, dry gas is mainly methane and wet gas contains ethane, propane, and butane.

After the formation of fluid hydrocarbons (oil and natural gas) in the source rock, they migrate from the source rock. There are two types of migration associated with petroleum: primary migration and secondary migration. Primary migration is the emigration of hydrocarbons from the source rock (clay or shale) in to reservoir rocks or permeable carrier beds,

usually sandstones or limestones. And secondary migration refers to the subsequent movement of oil and gas within the permeable carrier beds and reservoirs through the interconnected pore spaces of these rocks (Showalter, 1979; England, 1994). Primary migration may be partly due to the fact that petroleum in the source rock is under tremendous lithostatic pressure as a result of the weight of over-lying layers of rock and sediment. Secondary migration is by buoyancy as a result of the different densities of the fluids involved and in response to differential pressures (Selley, 1998).

2.2 Petroleum reservoirs and types of reservoir porosity

The most common types of reservoirs into which oil and gas migrates and are stored are sandstone reservoirs and carbonate reservoirs. The two major types of carbonate reservoirs are limestone reservoirs and dolostone (or dolomite) reservoirs. These reservoirs are porous and permeable. Selley (1998) describes the following porosities associated with petroleum reservoirs.

Primary porosity (depositional porosity) is the original porosity formed at the time of deposition. Primary porosity can be subclassified as intergranular porosity and intragranular porosity. Intergranular porosity occurs between grains and is initially present in all sediments at the time of deposition but can be quickly lost during lithification. The preserved forms of intergranular porosity are common in sandstone reservoirs. Intragranular porosity is found within grains especially within grains of carbonate sands.

Secondary porosity (post depositional porosity) is formed in the rock at some later time after deposition. Secondary porosity includes, intercrystaline porosity, fenestral porosity, vuggy porosity, cavernous porosity, moldic porosity, and fracture porosity. Moldic porosity is the

result of the selective dissolution of minerals, commonly carbonates, in which only the grains or only the matrix has been leached out. Vuggy porosity is also the result of the dissolution of minerals but cut across grains and matrix and is often larger than moldic pores. Larger forms of vuggy porosity are known as cavernous porosity. Fenestral porosity is characteristic of pelmicrite (carbonates) and is usually generated between buckled pelmicrite laminae. Dehydration, lithification etc soon after deposition can cause laminae to buckle and generate sub-horizontal fenestral pores between the laminae. Intercrystalline porosity is pores between the crystal faces of crystalline rocks. Fracture porosity refers to interconnected pores generated as fractures in brittle rocks.

Petroleum will migrate through the permeable reservoir rock until it encounters on its way, a highly impermeable rock unit called a cap rock which are commonly shale, gypsum, anhydrite, and salt (halite) which prevents further migration. When trapped, petroleum accumulates overtime in the reservoir rock (Selley, 1998; Murck et al., 1996).

2.3 Petroleum traps

A geologic setting involving a source rock, a reservoir rock, and a cap rock is known as a petroleum trap (Murck et al., 1996). There are different types of petroleum traps which can be classified into structural traps, stratigraphic traps, hydrodynamic traps and combination traps. Structural traps and stratigraphic traps are the major groups of traps agreed upon. Selley (1998) provides the following definitions of the different types of traps. Structural traps are those traps whose geometry was formed from tectonic processes after the deposition of the beds that formed them, resulting in folding or faulting. Stratigraphic traps are those traps whose geometry formed by depositional changes in lithology (e.g., channels and bars) or post depositional changes in

lithology (e.g., truncations and diagenetic changes). Hydrodynamic traps are those traps that occur where the downward movement of water prevents the upward movement of oil. Such traps are very uncommon. Combination traps are formed from a combination of any two or more of the above mentioned processes but very commonly from structural and stratigraphic processes. There are several specializations of the above mentioned traps such as fault trap and fold (anticlinal) trap related to structural traps. Also somehow related to structural traps are diapiric traps (salt diapirs and mud diapirs) which are caused by upward movement of impermeable materials such as salt or clay that are less dense than those overlying them. Unconformity traps, diagenetic traps, depositional traps such as pinchout traps, barrier bar traps are different kinds of the stratigraphic trap.

According to Selley (1998), an anticlinal trap is the simplest kind of trap and further provided the following definitions for its parameters. The productive reservoir within a petroleum trap is also known as the pay, and the vertical distance from the top of the reservoir to the petroleum/water contact is termed the gross pay, all of which may not necessarily consists of productive reservoir. The net pay is the cumulative vertical thickness of the reservoir from which petroleum may be produced.

2.4 Petroleum production and drive mechanisms

Production (or recovery) of oil and gas from the reservoir of a petroleum trap involves both natural and artificial drive mechanisms. Production of hydrocarbons may occur in stages identified as primary recovery, secondary recovery, and tertiary recovery (Enhanced Oil Recovery, EOR). Primary recovery (or primary production) involves natural drive mechanisms

of the reservoir in which oil and gas are produced using the reservoir's natural or original energy as a drive. Water drive, gas cap drive, dissolved gas (solution gas) drive, and gravity drive are primary (natural) recovery drive mechanisms. In water drive, as oil and gas are produced, the reservoir pressure drops and water occupying lower portions of the reservoir pushes the overlying oil upwards through the reservoir pores. In this case the drive energy is from an aquifer that in contact with the oil at the oil-water contact and displaces the oil during production (Selley, 1998). In gas cap drive, the drive energy for moving the oil is provided by the less dense natural gas which settles above the oil as a gas cap. As production continues resulting in the reduction of the reservoir pressure, natural gas in the gas cap expands and pushes the oil into production well. The drive energy in dissolved gas drive is provided by the dissolved gas in oil which bubbles out of solution and expands as the reservoir pressure drops during production. The expanded gas forces the oil through the reservoir pores towards the boreholes (Selley, 1998). Gravity drive involves the gravitational force of the earth pulling downward on oil towards production wells.

Secondary recovery and tertiary recovery involve artificial drive mechanisms. Artificial drive mechanisms involve supplemental effort to improve oil recovery when the natural drive of the reservoir becomes depleted. Secondary recovery techniques are water flooding (water injection) and gas flooding (gas injection). Water flooding involves the injection of water in to the reservoir to restore reservoir pressure and displace oil toward production wells. Gas flooding involves the injection of a gas into the reservoir to displace oil towards production well and to maintain gas cap pressure.

Tertiary recovery techniques include miscible gas injection, steam injection, detergent injection and alkaline flooding. Miscible gas injection involves the injection of gases such as carbon dioxide that are miscible with the oil to make oil more fluid and be drawn to production wells. Steam injection as the name implies involves the injection of steam into oil and gas reservoirs to make oil less viscous and be drawn into production wells. Some of the oil becomes vaporized. The steam upon cooling turns into water which also aid in driving oil to the well bore. The injection of detergents (micellar floods) is to emulsify heavy oil and move it to the surface (Selley, 1998). Similarly, alkaline floods reduce the interfacial tension between oil and water phases and thus enhance production.

CHAPTER 3

THE SWEET ONTOLOGIES

The Semantic Web for Earth and Environmental Terminology (SWEET) ontologies are developed and maintained by NASA's Jet Propulsion Lab. SWEET is a comprehensive upper-level ontology which comprises several thousand terms and related concepts drawn from the broad extent of Earth system science (Raskin and Pan, 2005). The ontologies are developed using the Web Ontology Language (OWL) and are downloadable at: http://sweet.jpl.nasa.gov/sweet (Raskin, 2005a). In the original SWEET ontologies, Figure 3.1, each rectangular box represents a separate ontology, and the connecting lines show where major properties relate these ontologies.

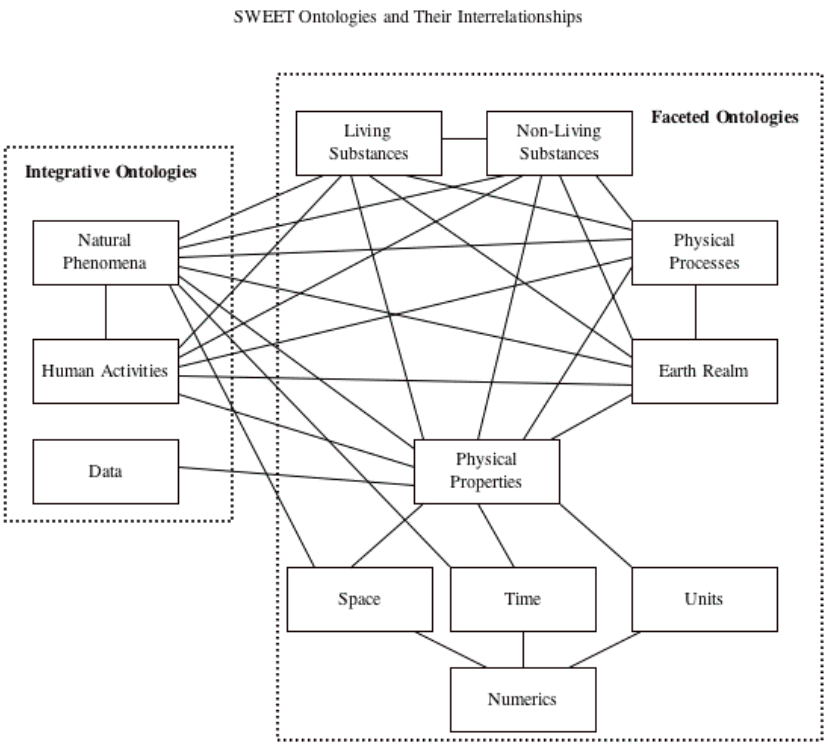

Figure 3.1 Primary ontologies in SWEET and their interrelationships (Raskin, 2005b).

The guide to the original SWEET ontologies (Raskin, 2005b) downloadable at: http://sweet.jpl.nasa.gov/guide.doc reveal that the primary ontologies in SWEET shown in Figure 3.1 are of two categories: integrative ontologies and faceted ontologies.

The faceted ontologies are surrounded by the larger dotted rectangle in Figure 3.1. The faceted ontologies represent orthogonal concepts commonly known as facets (faceted concepts). Facets are single concepts such as *fluid* and *pressure* which may exist as descendants of the same or different taxonomies (meaning they may be found as concepts in the same or different ontologies). The structure of a faceted ontology is designed such that specialization and greater details are added to more general concepts as one descends down the structure (or tree).

The integrative ontologies, surrounded by the smaller dotted rectangle in Figure 3.1, are unifying ontologies which define holistic, synthesizing concepts using elements from the faceted ontologies. The integrative ontologies contain integrative (unifying) concepts (Raskin, 2006). They may therefore define compound concepts such as *fluid pressure* which are combinations of orthogonal concepts.

Among the outstanding features of the SWEET ontologies are their scalability and orthogonality. Scalability is the attribute of ontology to be easily extendable so that specialized domain concepts can be developed from existing more general ontologies. Orthogonality of the SWEET ontologies implies compound concepts are decomposed (or split) into their component parts (facets) so that they can be recombined in diverse ways as desired. In orthogonal ontologies, each term (concept) is defined only in one ontology and other ontologies that need to use the term refer to its definition in the source ontology (Ghazvinian et al., 2010) to enhance term re-usability.

The current SWEET ontologies upon which the modifications presented in this thesis are made are downloadable at: http://sweet.jpl.nasa.gov/2.3. This current version of SWEET consists of nine top-level concepts/ontologies. These top-level ontologies consists of modules for (or cover areas such as) Representation, Process (microscale), Phenomena (macroscale), Matter, Realm, Human Activities, Property (observation), State (adjective, adverb) and Relation (verb). Relating to each module, there are several ontologies. For example, more directly related to the properties ontology are the *propChemical.owl*, *propEnergy.owl*, *propMass.owl* etc files all of which contain ontologies.

Representation has ontologies relating to Math, Science, Data, Time and Space. Again, several concepts are usually covered in the ontologies relating to each area. For example, *reprMath.owl, reprMathFunction.owl, reprMathOperation.owl,* and *reprMathStatistics.owl* are all files of representation relating to Math. They contain concepts such as Function, ExponentialGrowth, and Subtraction.

Process (microscale) has ontologies relating to Physical Process, Chemical Process, Biological Process and Mathematical Process. These ontologies consist of concepts including Diffraction, Reflection, Flushing, Folding, Flaring, Combustion, Trapping, Migration, Vaporization etc. Such processes affect, i.e, change the state of, both living and nonliving substances.

Phenomena (macroscale) consist of ontologies relating Ecological Phonomena and Physical Phenomena. It has related files such as *phenGeolTechnology.owl*, *phenHydro.owl* and *phenGeolFault.owl*. Concepts covered in this ontology include StrikeSlip, Normal, TectonicProcess, Subduction, and Spreading.

Matter consists of Living Thing, Chemical and Material Thing ontologies. Related files in this ontology include *matrNaturalResource.owl*, *matrCompound.owl*, *matrEnergy.owl*. The ontologies cover concepts such as FossilFuel, NaturalGas, Chlathrate, Biofuel, etc.

Realm consists of ontologies relating the layers of the Earth system (Earths' spheres) such as Atmosphere, Cryosphere, Geosphere, Heliosphere, Ocean, Terrestrial Hydrosphere, and Land Surface. Files such as *realmGeolOrogen.owl*, *realmGeolOceanic.owl*, *realmGeolConstituent.owl* are related to the realm ontology, and include concepts like Orogeny, Lithosphere, Matrix, and Landform.

HumanActivities ontology represents activities humans are engaged in. This embrace areas such as Human Decision, Commerce, Jurisdiction, Research and Environment and involves concepts such as Exploration, Prospecting, ResourceeExtraction, Imaging, Mapping, Drilling, Pumping, Mining etc.

Property (Observation) ontology relates Quantity (e.g., VectorQuantity), Binary Property (e.g., Polarity), Categorical Property (e.g., Group, Community, Classification), and Ordinal Property (e.g., Condition, Level, Color). Other concepts in the property ontology include physical properties such as permeability, porosity, thickness, altitude, depth, viscosity, density, and pressure. These properties are usually measured physical quantities or qualities with units.

State ontology includes Role, Physical, Chemical, Space and Biological state. It comprises files like *stateRealm.owl*, *statePhysical.owl*, and *stateChemical.owl*. These ontologies take account of concepts such as StateOfMatter, PhysicalState, Liquid, Gas etc.

Relation (verb) ontology covers areas such as Human, Physical, Time, Space, and Chemical relations. It incorporates the files *relaSci.owl*, *relaSpace.owl* and *relaTime.owl* Concepts such as DarcysLaw, Chronology, Provenance, Genesis etc are covered in these ontologies.

The relationships between concepts in the ontologies are usually that of a superclass-subclass relationship. Other relationships include disjointness and equivalence. Some classes or concepts were created by placing restrictions on the values of associated properties.

CHAPTER 4

CONTROLLED VOCABULARY AND ONTOLOGY

Controlled vocabularies are the set of approved terms such as unconformity, diagenesis, permeability, reservoir, orogeny, etc which are used in a domain of discourse as a means of communication. These terms are defined and expected to be consistent in meaning in all contexts (Hebeler et al., 2009). Vocabularies become controlled when they are managed, meaning when they are consensually approved and defined in which case a team of experts supervise (oversee) the addition (or creation) of new terms and revision of existing terms which are done according to agreed-upon procedures (Neiswender,2009). Glossary of terms (e.g., glossary of terms in Petroleum Geoscience), taxonomy, thesaurus etc are all examples of types of controlled vocabularies. The accepted terms are to follow a consistent definition and use. Within the framework of metadata (data which describes data), it is useful not to have multiple terms represent the same concept or the same term carry multiple meanings (i.e. the same term representing different conceps) to make it easier for data to be understood (Neiswender et. al, 2011).

Taxonomy is a kind of a controlled vocabulary in which terms are organized in the form of a hierarchy (Neiswender, 2009). According to Hebeler et al., (2009), whereas vocabularies (or controlled vocabularies) are a simple collection of well defined-terms, taxonomies extend controlled vocabularies by adding hierarchical relationships to the terms. The relationships between these terms are that of superclass-subclass relationship (Hebeler et al., 2009). Taxonomies and other examples of Knowledge Organization Systems (KOSs) such as thesaurus

can greatly help to formally organize knowledge of a given domain thereby enhancing reuse of the knowledge and also make possible or easy data (or metadata) interoperability (Yu, 2011).

Metadata interoperability is the ability of two or more computer systems to exchange (or share) descriptive metadata with minimal loss of information (Neiswender and Montgomery, 2009). For greater data interoperability, sophisticated data comparison, and improved knowledge discovery, controlled vocabularies alone are unable to meet these needs. This is where their use in an ontology satisfies the need through the added capabilities of the ontology (Alexander, 2011). A controlled vocabulary is viewed as an ontology when its concepts are explicitly defined and at least some defined as classes (Graybeal and Alexander, 2011). Also, a firm hierarchical subclass relationship must be present between the classes (Gruber, 1993). Graybeal and Alexander (2011), emphasized that to be adequately potent enough to be considered an ontology, a controlled vocabulary should possess classes, relations between classes, and properties. In this way, ontologies enrich controlled vocabularies by establishing relations between them and defining properties for terms (Alexander, 2011). An ontology uses predefined, reserved vocabulary of terms e.g., owl:class, owl:equivalentClass, rdfs:subClassOf, owl: allValuesFrom, etc to define concepts and the relationships between them (Hebeler et al., 2009). By this means, relationships between concepts will not only be human readable and understandable but machines (computers) will also be able to understand and interpret the encoded relationships between concepts. In this way the knowledge of a domain can be well encoded to be understood by a computer (Yu, 2011).

Further, though taxonomies and other Knowledge Organization Systems (KOSs) can be successfully used to organize knowledge, they cannot be used to represent knowledge. Also, because ontologies are based upon description logic it is possible to conduct logical inferencing

whereas taxonomies and other examples of KOSs relationships between concepts are semantically weak (Yu, 2011).

Ontologies can be very useful on the Semantic Web. The Semantic Web vision is to extend the current web so that data can be related or linked to one another and shared very efficiently based on meaning (semantics). On the Semantic Web information is represented as statements in the form of subject-predicate-objects, called triples (Hebeler et al, 2009). Information is encoded in triples using the Resource Description Frameworks (RDF) ontology language. The subject of the statement refers to the element being described, and the predicate describes the relationship between the subject and one or more object (Hebeler et al, 2009). For example, the statement: petroleum is contained in a reservoir rock, can be simply written as an RDF triple of the form, Petroleum containedIn ReservoirRock. The subject of this triple is Petroleum, the predicate is containedIn, and the object is ReservoirRock. With the RDF Schema, and to a large extent, the Web Ontology Language (OWL) semantics, or meanings, are added to the RDF statements. All resources (subjects, predicates, or objects) on the Semantic Web are identified using unique Uniform Resource Identifiers (URIs). A user with the appropriate query language, such as SPARQL, can request a URI and all the RDF triples where the URI dereferences to could be returned. In a more elaborate form, the request could be made in such a way that all the triples where the requested URI is a subject, predicate, or object could be returned. These resources can be referred to through namespaces or qualified names (qnames), which are used to qualify resources (terms) by associating the namespaces with URI references (Berners-Lee and Kagal, 2008).

CHAPTER 5

A BRIEF OVERVIEW OF THE WEB ONTOLOGY LANGUAGE (OWL)

The Web Ontology Language (OWL) (http://www.w3.org/TR/owl-features/) is a Semantic Web (Berners-Lee et.al, 2001) language for knowledge representation, and is based on Description Logic (DL) (Baader et.al, 2003), which allows a knowledge base, based on it, to respond with certainty to different kinds of data, models and queries (Babaie, 2011).

Web Ontology Language (OWL) extends the RDF(S) language (Horrocks et. al., 2003) by adding several constructs to those in RDF(S) language such, owl:unionOf, owl:disjointUnionOf, owl:intersectionOf etc. The Web Ontology Language (OWL) has several useful features. It has special features to make two classes, properties, and individuals equivalent. For example classes such as A and B can be made equivalent to each other using the owl:equivalentClass construct. Similarly two properties could be made equivalent using owl:equivalentProperty. Again in OWL we can constrain cardinalities using owl:Restriction to set a minimum cardinality as well as a maximum cardinality using owl:minCardinality and owl:maxCardinality respectively. Complex classes could also be constructed in OWL by restricting properties of existing classes. OWL defines two types of properties. OWL object property declared using owl:objectProperty and OWL datatype property, which is declared using owl:datatypeProperty. Object properties may have corresponding inverse properties. For example, an object property such as hasPart has the inverse, partOf. OWL Object properties also have the added value of property characteristics such as functional, inverse functional, transitive, symmetric, antisymmetric, reflexive, and irreflexive. Data properties may be only functional. Details of these properties are discussed later in this Chapter.

5.1 Defining classes

5.1.1 Property Restrictions

A property restriction results in the creation of a new class that is defined by the description of its members in terms of existing properties and classes (Allemang and Hendler, 2008). It describes the class of individuals that meet the condition or restriction placed on the property (Hebeler et al, 2009). For example, considering the property containedIn, if a restriction (or constraint) is placed on this property to have the value ReservoirRock as its object or range, this will define the class of things contained in a reservoir rock. A property restriction is therefore a special kind of class description. A property restriction is declared using the construct owl:restriction and the property on which the restriction is placed (i.e. the property the restriction applies to) is identified using owl:onProperty. So that in the above example the restriction defining the objects contained in a reservoir rock is owl:onProperty containedIn. The two types of property restrictions, value restrictions and the cardinality restrictions are discussed in the following sections.

5.1.2 Value Restrictions

A new class can be created or defined if the value of a property is restricted by specifying or stating what its object must be (Allemang and Hendler, 2008). A value restriction or constraint places constraints on the range of the property (Yu, 2011). For example considering the property hasChemicalElement, by restricting (constraining) the value (object or range) of this property to be HydroCarbon, we define a new class: the class of things that have chemical elements, Carbon and Hydrogen (Hydrocarbons). The three types of value restrictions used for

specifying the object or range of a property are owl:allValuesFrom, owl:someValuesFrom and owl:hasValue.

The owl:allValuesFrom restriction is used to specify that the value of the restricted property should all come from the specified class or data range (Yu, 2011). They are also known as 'only' restrictions as they constrain the range for a given property to a specific class (Horridge and Brandt, 2011) or data range. That means the value of the constraint property should be an instance of the specified class or data range. For example, the density of oil as mentioned earlier is often expressed in API gravity units (^{o} API) defined by the American Petroleum Institute (API). By the API definition, light oils have API gravities of more than 40^{o} and heavy oils have API gravities of less than 10^{o} (Selley, 1998). The above knowledge can be modeled using value constraints. By placing restriction on the property hasAPIGravity, and constraining its values to the specified range of GreaterThan40^{o}, we can define the LightOil class. Similarly the value (range) of the property hasAPIGravity can be constrained to LessThan10^{o} to define the HeavyOil class. Light oils have higher API gravities than heavy oils because oil viscosity and API gravity are generally inversely related (Selley, 1998). When the owl:allValuesFrom is used on a given property, the property does not need to have a value (no relationship for the property); the owl:allValuesFrom property restriction simply indicates that if a relationship exists with the given property, its values should all come from the specified class (Horridge and Brandt, 2011).

The owl:someValuesFrom restriction is used to define the class of individuals that have at least one specific kind of relationship, along the restricted property to individuals of the specified class (Horridge and Brandt, 2011). That means all instances (members or individuals) of the newly defined class must have at least one relationship along the restricted property to the specified range. For example, Petroleum is composed of hydrocarbon molecules and other

elements. To express this knowledge, a Petroleum class is defined by placing a constraint (or restriction) on the composedOf property to have owl:someValuesFrom the Hydrocarbon class, the range of the property. This means every individual of the petroleum class is at least composed of hydrocarbons. The owl:someValuesFrom restriction therefore provides some value restriction for the range of the property, in this case the Hydrocarbon class. Individuals of the defined class are not restricted to have values only from the specified range; they may additionally have a relationship along the property to individuals of a different class. For example, individuals of the defined Petroleum class apart from using range values from the specified Hydrocarbon class may additionally use the composedOf property with range values from a different class of elements. Petroleum may for instance be composed of other elements like sulphur, nitrogen, and oxygen (Hunt, 1996), which may belong to different classes of elements.

Similar to the earlier restrictions described, the owl:hasValue restriction acts on a particular property specified by owl:onProperty (Allemang and Hendler, 2008) and defines the class of individuals that are related to a specific individual (instance) along the specified property (Horridge and Brandt, 2011). It is nearly the same as the owl:someValuesFrom restriction except that the specified value or range of the particular property on which it acts is an instance (individual) instead of a class (Horridge and Brandt, 2011). This means that all instances of the defined class will have the particular property with the exact value specified. For example, to directly model the knowledge that wet gas contains significant amounts of condensate, the contains property when used with the WetGas class will be assigned the value SignificantCondensate as the range of the property. The class of oil fields located in Texas can also be defined using the locatedIn property which describes the class OilField, and a

specified value (owl:hasValue) of Texas. This provides a more direct approach to defining the class.

The code shown below uses the owl:someValuesFrom restriction in which Petroleum has been described to be composed of at least hydrocarbons. This is modelled using the composedOf object property, owl:someValuesFrom restriction, the Petroleum class, and the Hydrocarbon class.

Code using owl:someValuesFrom restriction:

```
<owl:Class rdf:about="&petronto;Petroleum">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&petronto;composedOf"/>
      <owl:someValuesFrom rdf:resource="&petronto;Hydrocarbons"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

5.1.3 Cardinality Restriction

Cardinality restriction is another way of defining a class by placing a constraint on the number of values a property can take (Yu, 2011). It makes it possible to define a class based on the number of distinct values a particular property of individuals of the class being defined can take (Allemang and Hendler, 2008). The cardinality restriction therefore indicates the number of times a property can be used to describe an instance of a class (Hebeler et al, 2009). Each individual of the class possesses the same number of distinct values of the property. The three

cardinality restrictions are: owl:cardinality, owl:minCardinality, and owl:maxCardinality. Assuming *x* is the number of distinct values of the property, (*x* being a non-negative integer) an owl:cardinality will have exactly *x* distinct values of the property. An owl:minCardinality will have at least *x* distinct values of the property and an owl:maxCardinality will have at most *x* distinct values of the property. The owl:minCardinality and the owl:maxCardinality can be used simultaneously to specify a range of values.

5.1.4 Using Set Operators

The set operators owl:intersectionOf, owl:unionOf and owl:complementOf can be used to define or construct more complex classes in terms of other existing simple classes.

For a class defined in terms of other classes using the owl:intersectionOf operator, the newly defined class becomes a subclass of each of the classes involved in its definition. For example if a SandstoneReservoir class is constructed from the owl:intersectionOf a Sandstone class and the PetroleumReservoir class, the SandstoneReservoir class will be a subclass of both the Sandstone class and the PetroleumReservoir class. Further, since a subclass will inherit all the properties of its base class (Yu, 2011), the SandstoneReservoir class will inherit all the properties associated with the base class Sandstone. Also, it will inherit all the properties associated with its other base class, which in this case is the PetroleumReservoir class.

A new class could also be defined by the union of earlier defined classes using the owl:unionOf operator. The list of classes which form part of the union in defining the new class will each become a subclass of the newly defined class (Yu, 2010). For example, a CombinationTrap (structural-stratigraphic kind) constructed from the union of StructuralTrap

and StratigraphicTrap will have both StructuralTrap and StratigraphicTrap become subclasses of the CombinationTrap. Some subclasses of the new class are purely or solely one of the participating components forming the union and other subclasses are intersections of any two or more of the participating classes forming the union (Yu, 2010). For example, the CombinationTrap mentioned earlier may have subclasses (sections) which are purely StructuralTrap, or purely StratigraphicTrap and a subclass which is actually an intersection of StructuralTrap and StratigraphicTrap. This means a combination trap of the structural-stratigraphic kind in an oil field may actually have sections which could be identified purely as a stratigraphic trap or a structural trap, and a trap involving both structural and stratigraphic processes (intersection). We construct the CombinationTrap from the union of its component parts because oil and gas fields that are due to a CombinationTrap may also have sections that are actually purely one of the component parts. However, strictly speaking, the CombinationTrap is that part of the trap that is formed from the intersection of its component parts. A Migmatite may also be similarly constructed from the union of the IgneousRock class and the MetamorphicRock class. This is because an outcrop of a migmatite will have sections which are purely metamorphic, and sections which are purely igneous, as well as sections which are both igneous and metamorphic.

The next set operator is the owl:complementOf. The complement of an OWL class will define another class whose members are things not in the complemented class (Allemang and Hendler, 2008). Such complement of a class will include all classes of individuals whose members are not in the complemented class. The owl:complementOf is often combined with the owl:intersectionOf operator to make the definition of a class complete and correct. For example, the complement of the SourGas class does not necessarily mean one is referring to the

SweetGas class. This is because the complement of the SourGas class includes the class of Rivers, Houses, Fruits, in fact the class of anything which is not SourGas. To indirectly assert that the SweetGas class is the complement of the SourGas class, the complement of the SourGas class (which could be the class of anything) is used in an intersection with the NaturalGas class. This is to assert that SweetGas is a NaturalGas. In other words, we are now considering a natural gas which is not sour, and therefore sweet. The definition is therefore made complete only after combining the owl:complementOf operator with the owl:intersectionOf operator. If a class, X is owl:complementOf of another class Y, then all the subclasses of the class X will be owl:disjoint With class Y (Yu, 2011).

In the code below, the SweetGas class is constructed from the intersection of the NaturalGas class and the complement of the SourGas class.

Code combining the owl:complementOf operator with the owl:intersectionOf operator:

```
<owl:Class rdf:about="&petronto;SweetGas">
    <owl:intersectionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="&petronto;NaturalGas"/>
        <owl:Class>
            <owl:complementOf rdf:resource="&petronto;SourGas"/>
        </owl:Class>
    </owl:intersectionOf>
</owl:Class>
```

5.1.5 Disjoint Classes

Two classes can be asserted to be disjoint when they do not have any individuals in common (Allemang and Hendler, 2008). In such a case, an instance of one class cannot at the same time also be an instance of the other class. Disjoint classes can be asserted using owl:disjointWith. The owl:disjointWith property is a symmetric property, that is if a class, X is disjoint with another class,Y then Y is also disjoint with X. For example, if the Oil class is disjoint with the NaturalGas class, it also implies the NaturalGas class is disjoint with the Oil class.

In cases where there are more than two classes, the owl:AllDisjointClasses OWL 2 operator can be used in a more concise way to specify that these classes are pair-wise disjoint with each other. Alternatively, OWL 1's owl:disjointWith has to be used on each pair of classes (Yu, 2011). For example, we can succinctly assert that the DissolvedGas, AssociatedGas, and NonAssociatedGas classes are pair-wise disjoint with each other using the owl:AllDisjointClasses.

The code below describes the AssociatedGas, DissolvedGas, and the NonAssociatedGas classes as all pair-wise disjoint classes:

```
<rdf:Description>
    <rdf:type rdf:resource="&owl;AllDisjointClasses"/>
     <owl:members rdf:parseType="Collection">
        <rdf:Description rdf:about="&petronto;AssociatedGas"/>
        <rdf:Description rdf:about="&petronto;DissolvedGas"/>
        <rdf:Description rdf:about="&petronto;NonAssociatedGas"/>
     </owl:members>
</rdf:Description>
```

Also, OWL 2's owl:disjointUnionOf, can be used in a concise manner to define a new class as the union of several collection of classes which are mutually or pair-wise disjoint with each other (Yu, 2011). For example, a petroleum trap in general consists of a source rock, a reservoir rock and a cap rock which are pair-wise disjoint with each other, meaning no component of the petroleum trap will for instance serve as a reservoir rock and a cap rock at the same time. The PetroleumTrap class can therefore be modeled as the owl:disjointUnionOf the SourceRock, ReservoirRock, and CapRock classes. Again, this implies there are no intersections of the classes involved in the union (or no members in the intersections). The classes are disjoint.

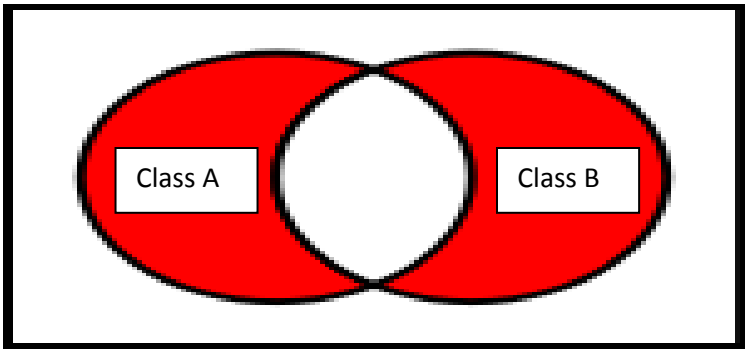

Figure 5.1. The disjoint union of class A and Class B shaded.

Code for constructing the PetroleumTrap class from the disjoint union of the SourceRock, ReservoirRock, and CapRock is shown below:

```
<owl:Class rdf:about="&petronto;PetroleumTrap">
    <owl:disjointUnionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="&petronto;CapRock"/>
        <rdf:Description rdf:about="&petronto;ReservoirRock"/>
        <rdf:Description rdf:about="&petronto;SourceRock"/>
    </owl:disjointUnionOf>
</owl:Class>
```

A similar construct can be used for modeling the Maturation process of organic matter as the owl:disjointUnionOf of Diagenesis, Catagenesis, and Metagenesis. The code for this modeling is provided below:

```
<owl:Class rdf:about="&petronto;Maturation">
    <owl:disjointUnionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="&petronto;Catagenesis"/>
        <rdf:Description rdf:about="&petronto;Diagenesis"/>
        <rdf:Description rdf:about="&petronto;Metagenesis"/>
    </owl:disjointUnionOf>
</owl:Class>
```

5.2 OWL Properties

In OWL, two major types of properties can be defined. They are declared using owl:ObjectProperty and owl:DatatypeProperty. The owl:ObjectProperty is used to connect a resource to another resource (Yu, 2011). As a binary property, the owl:ObjectProperty relates the set of individuals of a class to the set of individuals of another class. The owl:DatatypeProperty on the otherhand connects a resource to an rdfs:literal (untyped) or XSD datatype (typed) value (Yu, 2011). Also as a binary relation, the owl:DatatypeProperty relates the set of individuals (instances) of a class to the set of individuals of a datatype.

The definitions of properties can be enriched (enhance semantics) through the use of features or property characteristics. The characteristics that object properties may have are functional, inverse functional, transitive, symmetric, asymmetric, reflexive and irreflexive. Some of these property characteristics are discussed in the following section.

5.2.1 Inverse Properties

A property may be asserted to be the inverse of another property using owl:inverseOf. Inverse properties are associated with object properties. Each object property in one direction may have a corresponding inverse property in the opposite direction so that if the property relates individual x to individual y, then its inverse property will relate individual y to individual x (Horridge and Brandt, 2011). For example, the property hasPart has inverse partOf. For instance, if a PetroleumTrap hasPart ReservoirRock, then the ReservoirRock is partof the PertoleumTrap. Similarly, if in petroleum production, PrimaryRecovery hasDriveMechanism NaturalDrive, we can also say, NaturalDrive is the driveMechanismOf PrimaryRecovery. As said earlier, inverse properties are associated with only object

properties. Datatype properties cannot have an inverse because their values, which are literals, cannot be the subjects of statements (Hebeler et al., 2009).

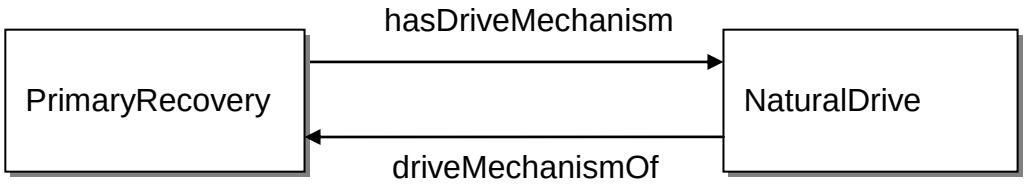

Figure 5.2. The hasDriveMechanism and driveMechanismOf inverse properties.

5.2.2 Property Characteristics

5.2.2.1 Symmetric Properties

A property P is declared symmetric using owl:SymmetricProperty if it describes a situation where it connects (or relates) a resource x to a resource y, and it again (the same property name) relates the resource y back to the resource x. The relationship is in both directions. The inContactWith property is an example of a symmetric property. For example, if a ReservoirRock is inContactWith a CapRock in a petroleum trap, the CapRock is also inContactWith the ReservoirRock. The property inContactWith is therefore a symmetric property.

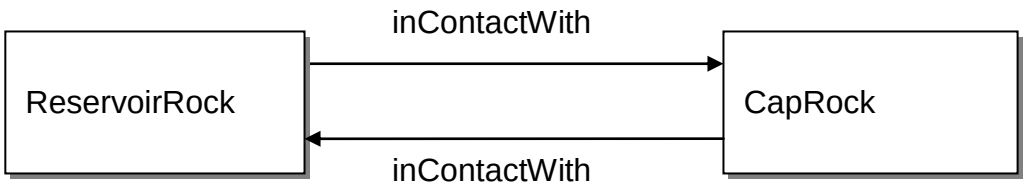

Figure 5.3. The inContactWith symmetric property

The rdfs:range of a symmetric property is always a resource (can only be a resource) and never a literal or datatype (Yu, 2011). This is because owl:SymmetricProperty is a subclass of owl:objectProperty, which as mentioned earlier relates a resource to another resource.

5.2.2.2 Asymmetric Properties

A property is an asymmetric property if it relates an individual x to individual y but does not relate individual y back to individual x. That means the relationship is not bidirectional. The migratesFrom property is an example of an asymmetric property. For instance if Oil migratesFrom SourceRock, the reverse statement which reads: SourceRock migratesFrom Oil does not hold true. The asymmetric migratesFrom property is able to relate oil to source rock in one direction but unable to relate source rock back to oil in the opposite direction. A property is declared asymmetric using owl:AsymmetricProperty.

5.2.2.3 Transitive Properties.

For individuals x, y, and z and a transitive property P, declared using owl:TransitiveProperty, (xPy) and (yPz) implies (xPz) (Hebeler et al, 2009). This means if a property P is transitive, and the property P relates individual x to individual y (xPy), and also relates individual y to individual z (yPz), then it can be inferred that individual x is related to individual z by the property P (xPz) (Horridge and Brandt, 2011). For example, the containedIn property can be declared transitive as explained below. Oil is contained in the pore spaces of a reservoir rock. And since the pore spaces are themselves contained in the reservoir rock, it follows transitively that oil is contained in the reservoir rock. Or better still, oil or gas fields

may contain one or more pools. Each pool may also contain one or more pay zones. We may therefore conclude that an oil or gas field may contain one or more pay zones.

The locatedIn property could be used in a similar sense. For example, if a particular Oil field is located in a certain county and that county is located in a given State, it implies transitively that this Oil field is located in the State under consideration. Transitive properties are also associated with only object properties. Datatype properties can not be declared transitive.

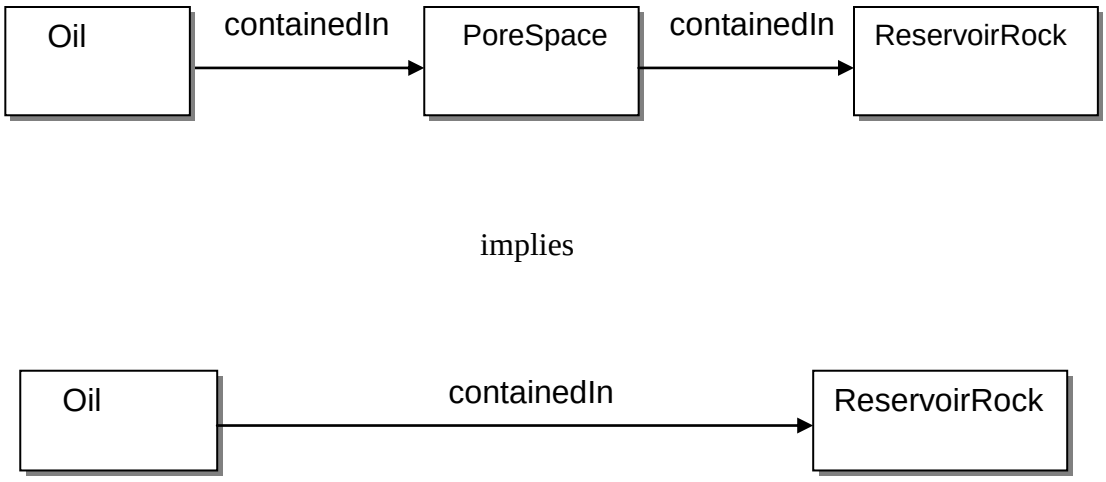

Figure 5.4. The containedIn transitive property

Like the symmetric property, owl:TransitiveProperty is a subclass of owl:objectProperty, therefore its rdfs:range can only be a resource but not a literal or datatype. It can only connect a resource to another resource (Yu, 2011).

5.2.2.4 Functional Properties

A functional property is one for which there is only one value for a given instance. It has one unique rdfs:range value for each rdfs:domain instance. It is declared using owl:FunctionalProperty, and it is a subproperty of the rdf:property. As a result, the rdfs:range of a functional property can be a resource, a literal, or a datatype (Yu, 2011).

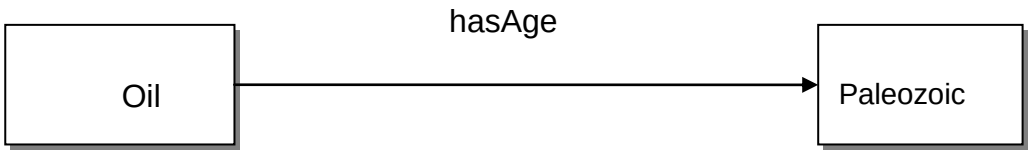

Figure 5.5 The hasAge functional property

Again, both an owl:ObjectProperty and an owl:DatatypeProperty can be functional, hence the rdfs:range values of the functional property can be any type of values that an owl:ObjectProperty or an owl:DatatypeProperty can assume. Properties such as hasLocation, hasArealExtent, hasSpecicificGravity, hasAge etc are properties which can assume only one value for any instance and are therefore characterized as functional properties. Further, if the property of a class is assigned an owl:cardinality equal to 1, it is the same as declaring that the property is a functional property (Yu, 2011).

5.2.2.5 Inverse Functional Property

The object individual x of an owl:InverseFunctionalProperty, uniquely relates (or identifies) a single subject individual y (Hebeler et al, 2009).

The inverse functional property is the opposite of the functional property. For a given rdfs:range value of a property, a unique rdfs:domain is associated with the property. Like the owl:FunctionalProperty, the owl:InverseFunctionalProperty is a subclass of the rdf:Property, as such its rdfs:range can be a resource, a literal, or a datatype (Yu, 2011). It is possible for a property to be functional but not inverse functional. For example, the source rock of a petroleum trap may have only one value for its density (measured in metric tons / cubic metre). The hasDensity property is therefore functional. However, this value of density does not uniquely identify this source rock under consideration. There could be other source rocks (even of other kinds) from different petroleum traps of the same value of density. The hasDensity property is therefore functional but not inverse functional. Some functional properties may however be inverse functional at the same time. This means they are functional, and their inverse is also functional. For example, a chemical element may always have a unique atomic number. The hasAtomicNumber property is therefore functional. Also with a given atomic number, only one particular element will always be identified. The inverse of the hasAtomicNumber property, which is, isAtomicNumberOf is also functional. The hasAtomicNumber property is therefore both functional and inverse functional.

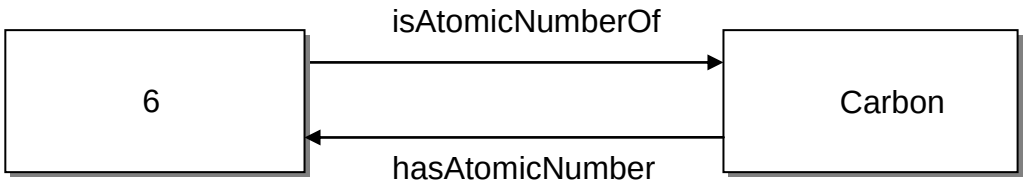

Figure 5.6. The hasAtomicNumber inverse functional property

5.2.3 Disjoint Properties

Two properties are said to be disjoint when no two statements can exist where the subjects and objects of those statements are the same and each of the two statements have one of the disjoint properties as its predicate (Hebeler et al., 2009). In other words, the two statements in which the disjoint properties serve as predicates cannot have the same subjects and the same objects; they may have the same subjects but different objects or they may have different subjects with the same objects. Owl:propertyDisjointWith construct is used to specify that two properties are disjoint . In cases of more than two properties Owl:AllDisjointProperties is used to specify that sets of the properties are pair-wise disjoint (Hebeler et al., 2009).

5.2.4 Domain and Range of Properties.

When a property is defined, rdfs:domain is usually used to specify which class the property can be used to describe. As a result, whenever a property is used to describe any resource, that resource is recognized by any application (by inferencing) as an instance of the class specified by the rdfs:domain. Similarly, rdfs:range is usually used to specify the values a property can assume. These values can themselves be instances of a declared class or can be literals which are directly typed or even un-typed (Yu, 2011). The values of a range which are instances of a class are associated with object properties and those values which are typed or literals (e.g., XSD or RDF literal) are associated with datatype properties. Domain and range therefore indicate how a property can be used (Allemang and Hendler, 2008).

If the domain of a property is not specified, that means the property can be used to describe any class. Similarly an unspecified range of a property suggests that the property can take on any kind of value. Multiple rdfs:domain can be used to specify that the resource is an

instance of all the classes defined by the rdfs:domain (Yu, 2011). For example, consider the code:

```
<owl:ObjectProperty rdf:about="&petronto;containedIn">
    <rdfs:domain rdf:resource="&petronto;DissolvedGas"/>
    <rdfs:domain rdf:resource="&petronto;SweetGas"/>
    <rdfs:range rdf:resource="&petronto;ReservoirRock"/>
</owl:ObjectProperty>
```

The above code suggests that the property containedIn can be used on instances that are DissolvedGas and SweetGas at the same time (and on only instances of these two classes specified by the rdfs:domain. A similar situation applies to the rdfs:range. In the example shown above, the range in the code applies to the ReservoirRock.

Further, a subclass will inherit all the properties of its base class (Yu, 2011). For example, if the class Petroleum is defined to have two disjoint classes Oil and NaturalGas as subclasses, all the properties associated with Petroleum will be inherited by Oil and NaturalGas. As a result, if the property containedIn for instance, is assigned to Petroleum and a statement is made to the effect that Petroleum is containedIn ReservoirRock as shown below:

```
<owl:ObjectProperty rdf:about="&petronto;containedIn">
    <rdfs:domain rdf:resource="&petronto;Petroleum"/>
    <rdfs:range rdf:resource="&petronto;ReservoirRock"/>
</owl:ObjectProperty>
```

The property, containedIn is inherited by Oil and NaturalGas (the asserted subclasses of Petroleum) and the implication of the above statement is that Oil is containedIn ReservoirRock, or NaturalGas is containedIn ReservoirRock, or both Oil and NaturalGas are containedIn ReservoirRock. If the value of the property (containedIn) is not specified (as ReservoirRock in this case), and left open as say a string, then the subclasses of Petroleum (the subject) will inherit the property (containedIn) and its value will be any kind of string. The subclasses of the base class could use the inherited property used in describing the base class with range values of any kind (Yu, 2011).

A property could also be defined as a sub-property of another property using the rdfs:subPropertyOf construct. Sub-properties inherit the domain and range values of their base properties (Yu, 2011; Babaie, 2011). For example, the sub-properties of porosity: vuggyPorosity, cavernousPorosity, fenestralPorosity etc will inherit the domain and range of porosity in any statement in which porosity is used as a predicate.

Properties are not unique to any class, and are not owned by any class. They are declared independently or separately and can be associated with any class where appropriate (Yu, 2011).

CHAPTER 6

METHOD

6.1 Reengineering the SWEET ontologies for Petroleum Geology

Ontological reengineering is the process of retrieving and transforming a conceptual model of an existing and implemented ontology into a new, more correct and more complete conceptual model which is reimplemented (Gomez-Perez and Rojas-Amaya, 1999). The process of reengineering used primarily in software engineering involves three major activities: *reverse engineering*, *restructuring*, and *forward engineering* (Gomez-Perez and Rojas-Amaya, 1999).

Chikofsky and Cross (1990) defined *reverse engineering* as the process of analyzing or examining a subject system (e.g., software program) to identify its components and interactions, and to create representations of the system at a higher level of abstraction. According to Warden (1992), *reverse engineering* can be seen as "going backwards through the development cycle of a system". Blum (1992) mentions that reverse engineering signify the process of analyzing a system to discover its make-up (components) and relations that exist between them and the representation of the system in another form. In *reverse engineering* therefore, no change or modification is done to the system; however discoveries are made upon which higher models of the system would be developed.

In this reengineering process a completely new ontology is built making use of concepts in the SWEET ontology as upper-level concepts, which are extended to construct the new ontology. First, identification and collection of important concepts (key concepts) and their consistent definitions as well as any relevant information about the Petroleum Geology domain of interest was made (Babaie et al., 2006). The relevant domain knowledge for this work was

acquired mainly through looking up existing glossary of terms in Petroleum Geology texts (which represents consensual knowledge), and also by interviewing experts in this domain. Hierarchies or taxonomic relations between concepts were then drawn.

The current SWEET ontologies (OWL files) were then accessed and examined for extension to suit the Petroleum Geology domain. Taxonomic relations between concepts in this ontology were also inspected and drawn. The next stage which is *restructuring* is defined by Chikofsky and Cross (1990) as the transformation of a system from one representation to another at the same level of abstraction, maintaining the semantic behavior and functionality of the original system in the new. Gomez-Perez and Rojas-Amaya (1999) identified two phases in *restructuring:* analysis and synthesis. The analysis phase involved a technical evaluation of the ontologies in which the ontologies were checked for correctness and completeness of hierarchies (the taxonomic relations between concepts). Concepts in the SWEET ontology were then used as upper-level concepts in constructing a new ontology. At this point any missing concepts required were added to the new ontology. Also, the correctness and completeness of the definitions of classes and properties were checked. The scope of the extended ontology is best determined by putting forward a list of motivating scenarios and competency questions which we seek the ontology to address (Gruniger and Fox, 1995). A new ontology is then developed (synthesis phase) which outputs the design of a new conceptual model bearing the correctness and completeness of hierarchies and definitions of concepts (Gomez-Perez and Rojas-Amaya, 1999). In the *forward engineering* process the newly designed conceptual model is re-implemented. It may then be more suitable to be used in different respects, especially for the purpose for which it was re-engineered.

6.2 Developing the ontology using Protégé OWL Plugin

Having collected the relevant domain concepts and established the desired superclass-subclass relationships between concepts, and the appropriate properties for linking concepts, the designed ontology is developed in Protégé. The Protege 4.1 beta used in developing or building this ontology is fully conformant with the OWL 2.0 language specifications, which is a W3C recommendation as of October 27, 2009. The Protégé OWL Plugin is an extension of the Protégé (Gennari et. al., 2003) ontology development environment. The OWL Plugin extends protégé to support the Web Ontology Language (OWL). The Protégé OWL Plugin can therefore be used to load and save OWL files in various formats, define classes and properties, edit ontologies in OWL, use reasoners (like FaCT++, HermiT, etc) and invoke Description Logic (DL) reasoning such as consistency checking and classification (Knublauch et. al., 2004). Description Logic (DL) reasoners help in building and maintaining ontologies by not only revealing inconsistencies and hidden dependencies, but also revealing redundancies, and misclassification (Knublauch et. al., 2004).

The protégé OWL Plugin comes with several features. Its user interface consists of tabs (OWL Classes tab, Properties tab, Individuals tab etc) which display different aspects of the ontology. In general, each tab consists of two main sections. The left section of both the classes and properties tabs show hierarchy of objects (classes, properties). The right section of the tabs displays details of a selected object.

6.2.1 The OWL Classes tab

With the Protégé's OWL Classes tab, classes can be created and edited. The Classes tab also displays the class hierarchy of the ontology. Figure 6.1 is a screenshot of the OWL Classes tab. The main class hierarchy appears on the left section of the screen. The detail of the class that is currently selected appears in a form on the right section of the screen. The upper section of the class form shows annotations whiles the lower section shows information about equivalent classes, superclasses, inherited anonymous classes and disjoint classes of the selected class. The "Add subclass", "Add siblings" and "Delete selected class" buttons (icons) are shown.

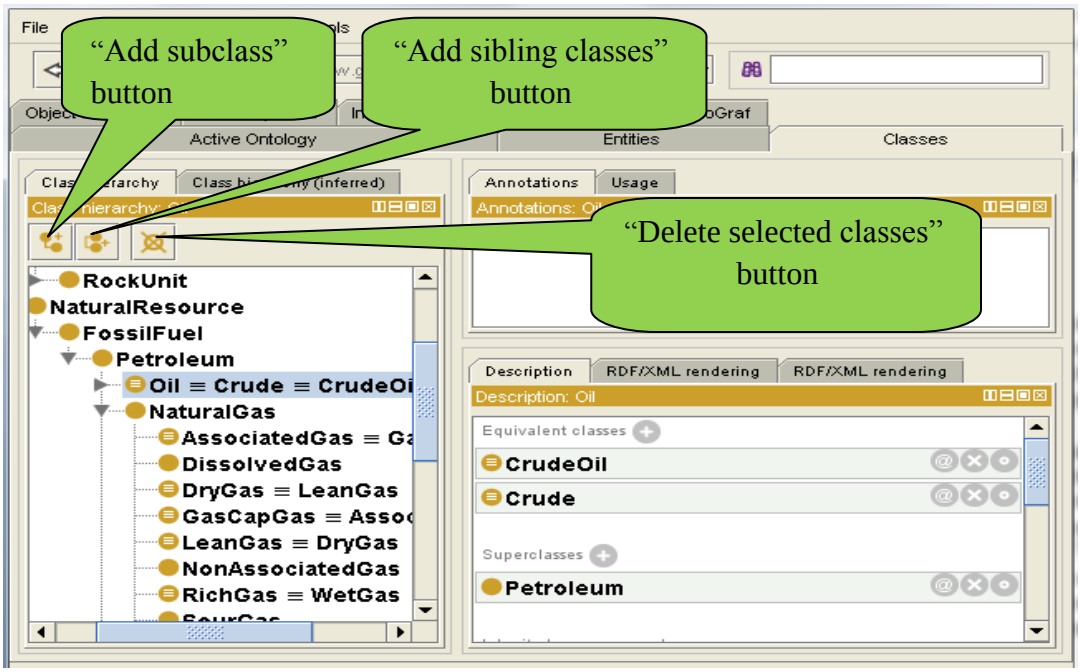

Figure 6.1. Screenshot of the Protégé OWL Classes tab for creating classes.

Basic classes such as Petroleum, Oil, NaturalGas, PetroleumTrap, etc, are created by clicking the Classes tab and then selecting "Thing". In OWL, owl:Thing is the root class of all classes, therefore every class is created as a subclass of the "Thing" class. After selecting

Thing, click on the "Add subclass" button. A dialog box will appear for the name of the class to be entered as shown in Figure 6.2. Then click OK to finish naming the class. The naming convention followed in this thesis is to start the class names in uppercase and Arial font e.g., Petroleum. Where a class name is composed of more than one word, it is concatenated by starting each word with an upper case letter e.g., NaturalGas or PetroleumTrap.

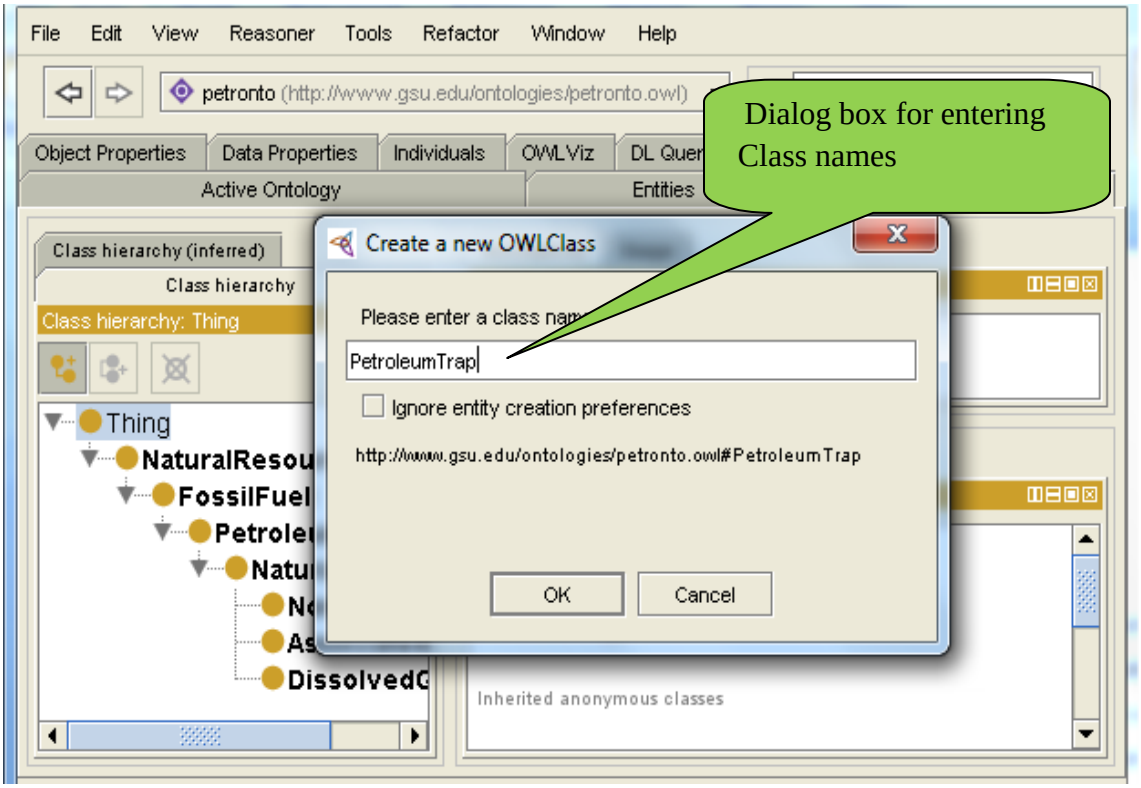

Figure 6.2. Screen shot showing pop up dialog box for entering class names to be created. In this screen shot the class PetroleumTrap is being created.

To create a subclass of any named class, for example to create Oil as a subclass of Petroleum, we select Petroleum, click on the "Add subclass" button, and type the name of the new class, Oil in the opened dialog box and click OK, just as described above for creating

subclasses of "Thing" class. Figure 6.3 shows a screen shot of creating subclasses. The screen shot shows the selection of the Petroleum class and the "Add subclasses" button to be clicked to create subclasses of Petroleum such as CrudeOil and NaturalGas.

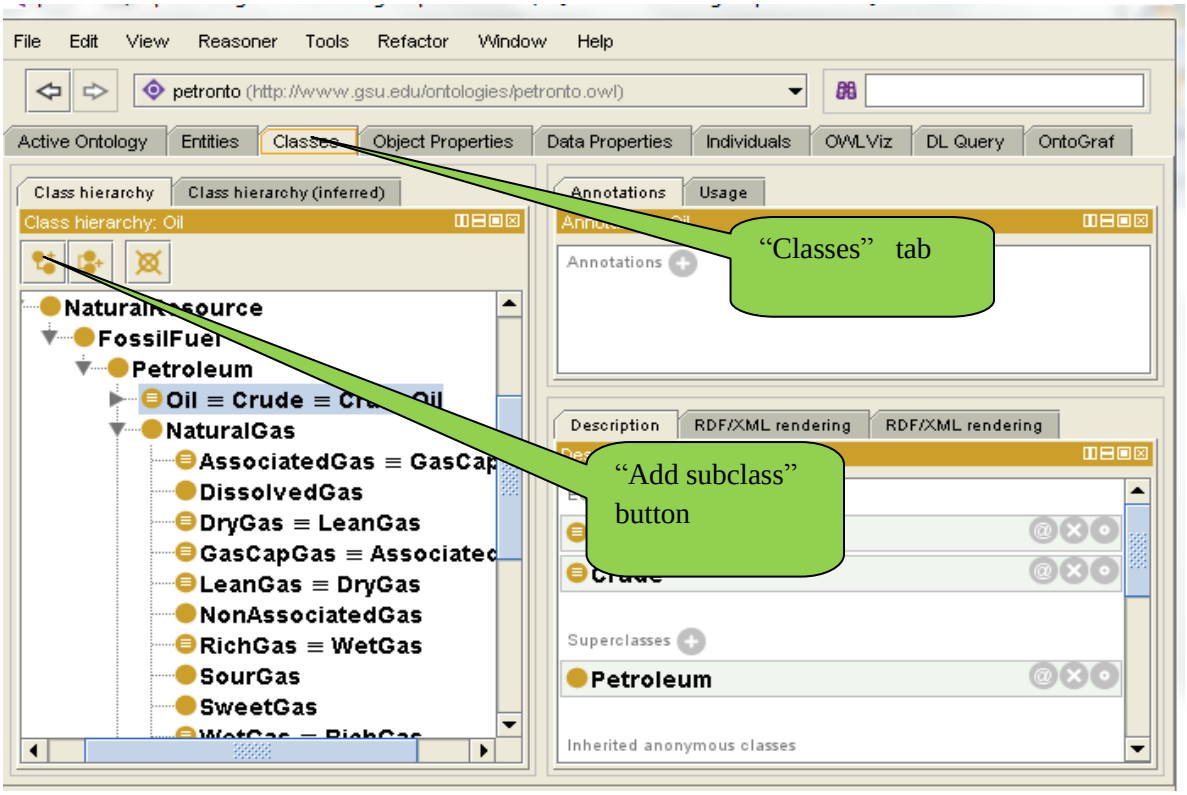

Figure 6.3. A screen shot showing the "Classes tab" and the "Add subclass" button for creating subclasses.

Similarly, to create siblings, for example to create DryGas as a sibling of WetGas, we first select the named class whose sibling we seek to create, i.e. WetGas, then click the "Add sibling class" button. Again a dialog box will appear into which the sibling class, DryGas is typed, following the naming convention mentioned earlier. Then click OK to finish. Any created class can be deleted by selecting the class and then clicking on the "Delete selected classes" button adjacent to the "Add sibling class" button.

Two asserted classes such as SweetGas and SourGas can be made disjoint in Protégé by selecting one of the classes (e.g., SweetGas), then clicking on the "Add" icon by the "Disjoint classes" in the "Description" view. A window pops up showing the class hierarchy from which the other class to be made disjoint (SourGas) is selected in this case. You then press OK to complete the process.

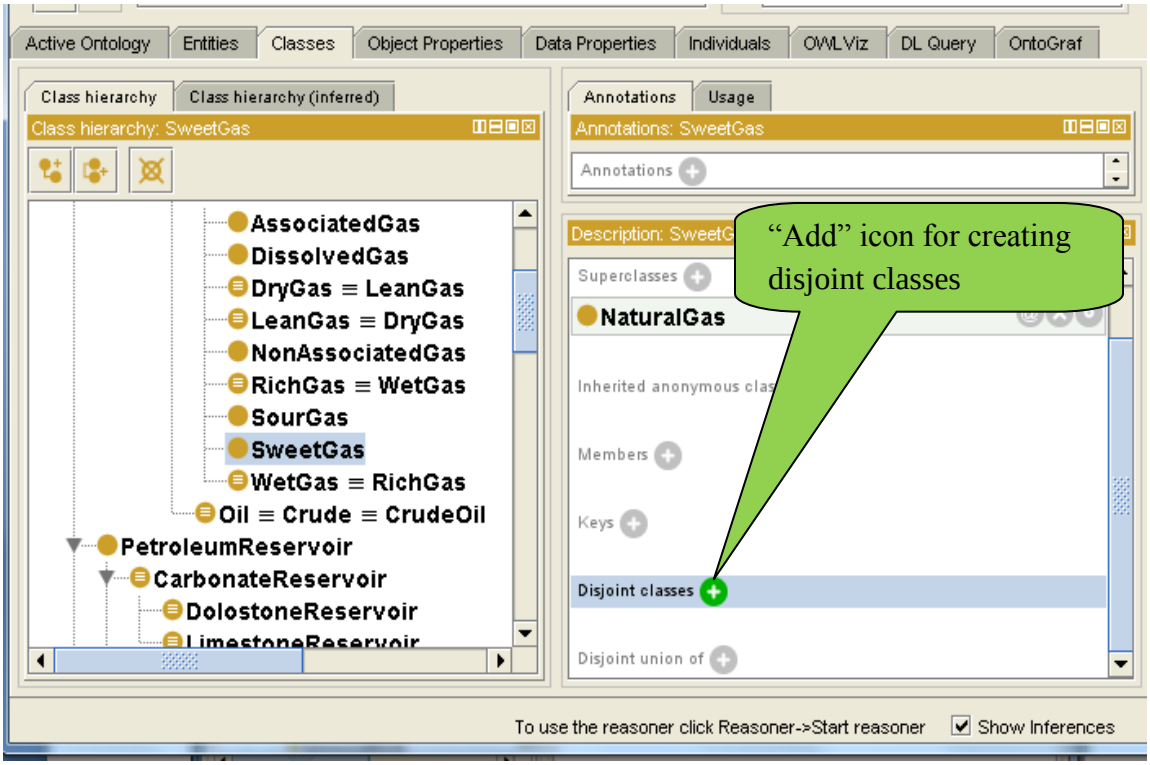

Figure 6.4. Screen shot showing the creation of disjoint classes using the "Add" icon by the "Disjoint classes". SweetGas is selected to be made disjoint from another class.

Several asserted classes can made pair-wise disjoint with each other in a similar manner but concisely without selectively creating them pair-wise in turns. As a simple example, to make the AssociatedGas, NonAssociatedGas, and DissolvedGas classes pair-wise disjoint with each other, select one of the classes e.g., AssociatedGas from the class hierarchy, then click on

the “Add” icon by the “Disjoint classes” just as done previously. A window with the asserted class hierarchies pops up, from which NonAssociatedGas is selected, and with the control key (Ctrl) held down the DissolvedGas class, the second class to be made disjoint is simultaneously selected, and then the OK button is clicked. With the control key held down, multiple classes can be assigned as disjoint to the selected class.

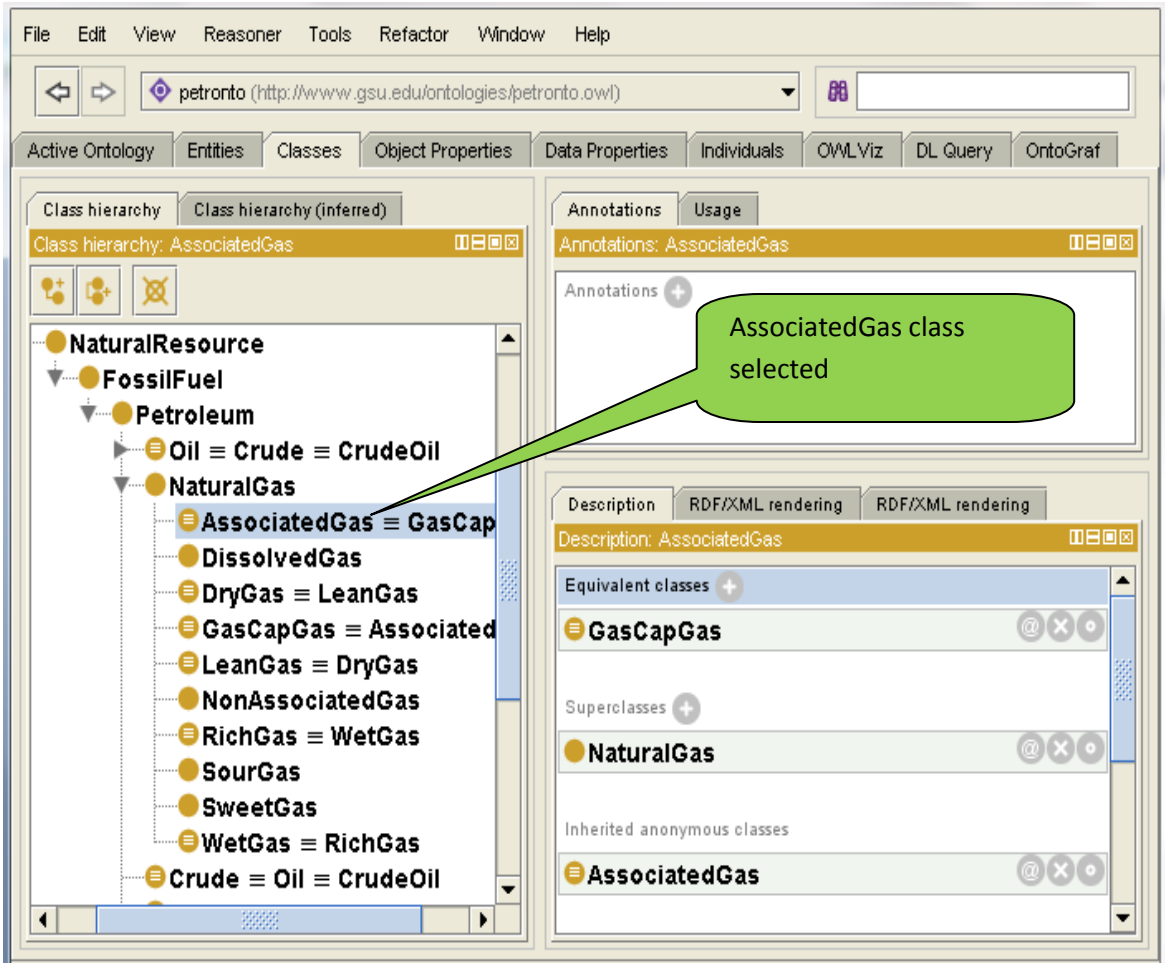

Figure 6.5. Screen shot showing the selection of AssociatedGas to create disjoint classes.

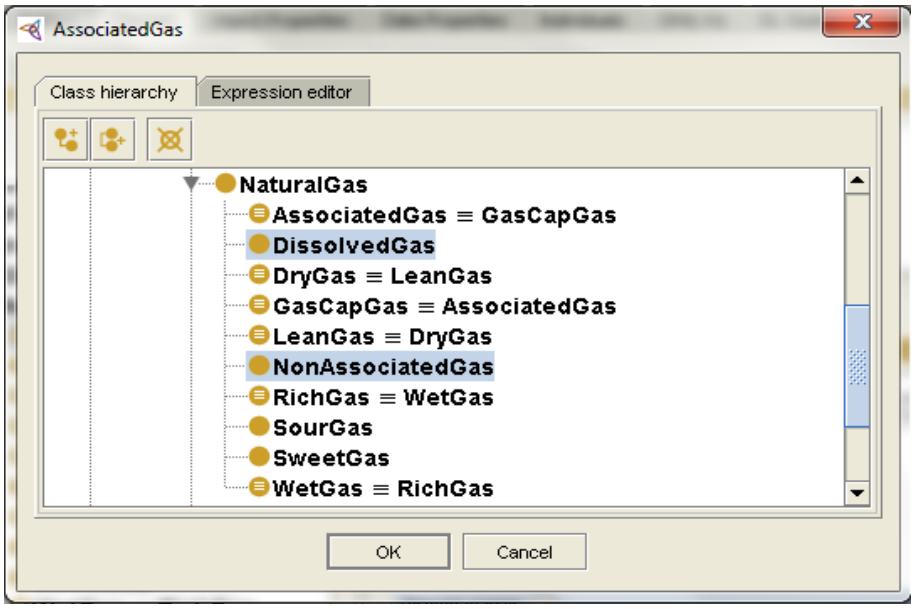

Figure 6.6. Screen shot showing the selection of multiple classes to be made pair- wise disjoint with the earlier selected class in Figure 6.5

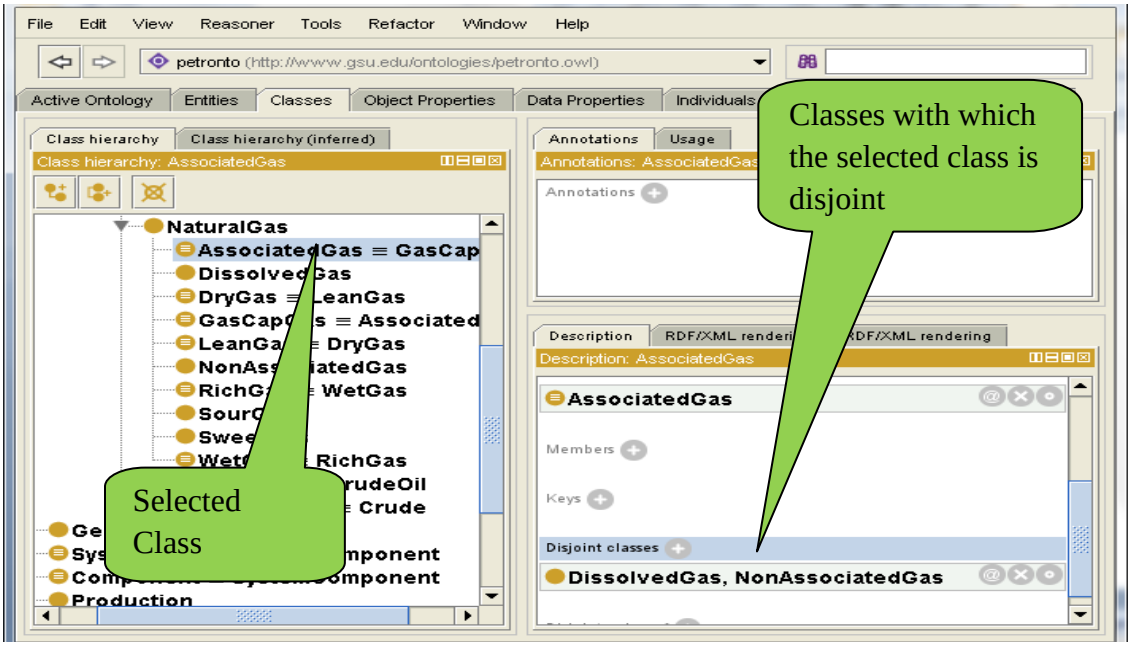

Figure 6.7. Screen shot showing a selected class, and its multiple pair-wise disjoint classes.

Complex classes such as SandstoneReservoir, PetroleumTrap, CombinationTrap, etc were constructed using set operators. A SandstoneReservoir for example, is a sandstone

and also a petroleum reservoir. The SandstoneReservoir class can be modeled by the intersection of the Sandstone class and the PetroleumReservoir class. In protégé, this modeling is done by first selecting the SandstoneReservoir class in the class hierarchy, then under the "Equivalent classes" section, a click on the "Add" icon will pop up a dialogue box. Using the "class expression editor" of the dialogue box, type in the participating classes in the intersection, separated by "and". In this example we type in the "class expression editor" Sandstone and PetroleumReservoir. We then click on OK to complete the process.

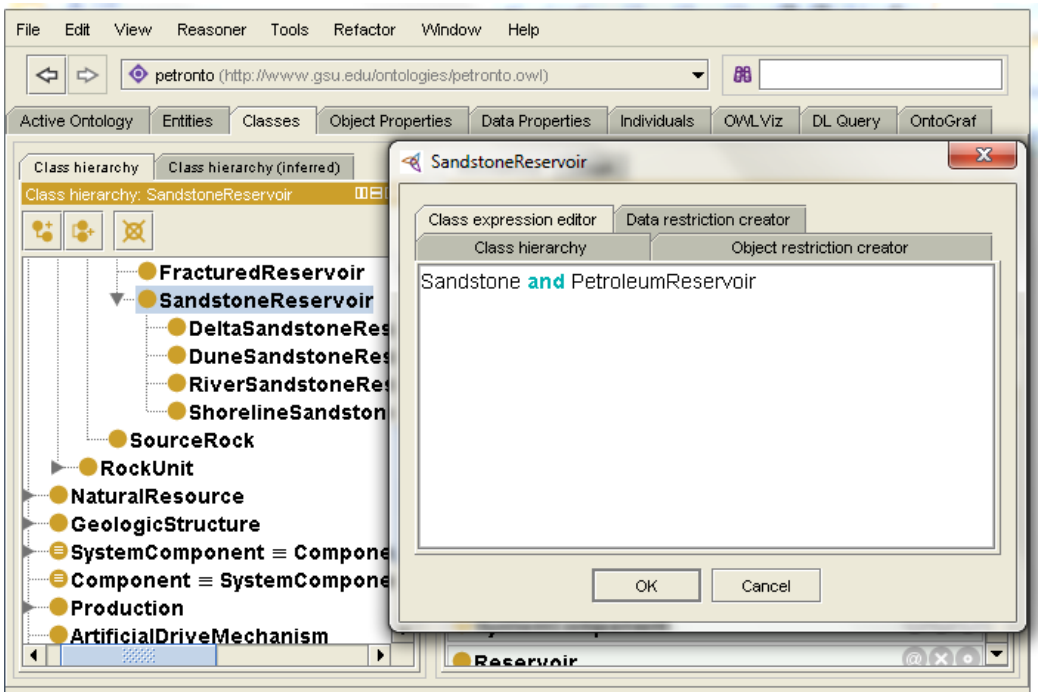

Figure 6.8. Screen shot showing the creation of a class from the intersection of two other classes.

A CombinationTrap of the structural-stratigraphic kind (by far the most common) is constructed from the union of a StructuralTrap and StratigraphicTrap. This is because geologically, an oil field consisting of such a trap may have sections that are formed through

purely structural processes, and sections formed purely through stratigraphic processes, as well as major sections that are formed through both structural and stratigraphic processes. The StructuralTrap and StratigraphicTrap classes are not made disjoint, to make provision for their overlap in the union. This class is created in protégé by first selecting CombinationTrap in the class hierarchy, and then under "Equivalent classes" section of the "Description view", it is defined as StructuralTrap or StratigraphicTrap. The definition is made by clicking on the "Add" icon next to the "Equivalent classes" section. A dialogue box pops up and using the class expression editor, type StructuralTrap or StratigraphicTrap, then press OK as shown in the figure below. Notice we use "or" to specify the union of the two classes.

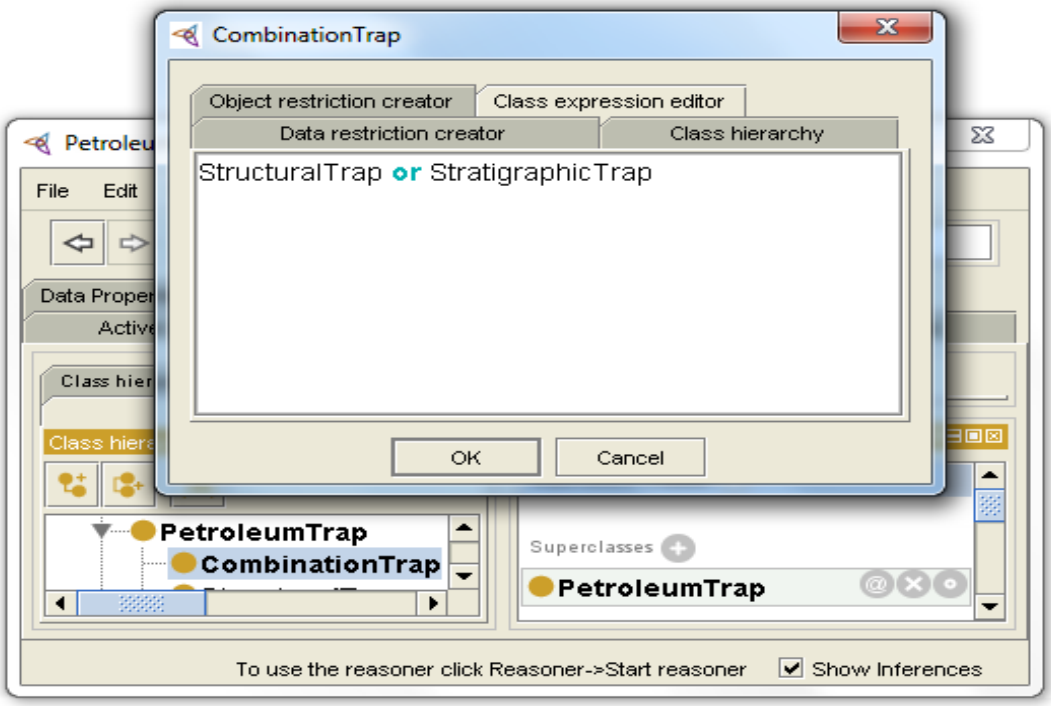

Figure 6.9. Screenshot showing the definition of a CombinationTrap as the union of the StructuralTrap and StratigraphicTrap.

A PetroleumTrap is constructed in a similar manner but using the disjoint union of the SourceRock, ReservoirRock, and CapRock, which petroleum trap generally consists of. The difference between the union, and the disjoint union is that, unlike the union, the disjoint union has no intersection of the participating classes (i.e., the intersection of disjoint classes is empty or null). In a geological sense, this will for instance mean that the CapRock will not also serve simultaneously as a ReservoirRock. In protégé this modeling is done by selecting PetroleumTrap in the class hierarchy and then clicking on the "Add" icon next to the "Disjoint union of" section of the "Description view", shown in Figure 6.10. A dialogue box will pop up from which the class hierarchy can be accessed and all the participating classes in the disjoint union selected with the aid of the control key (Ctrl key).

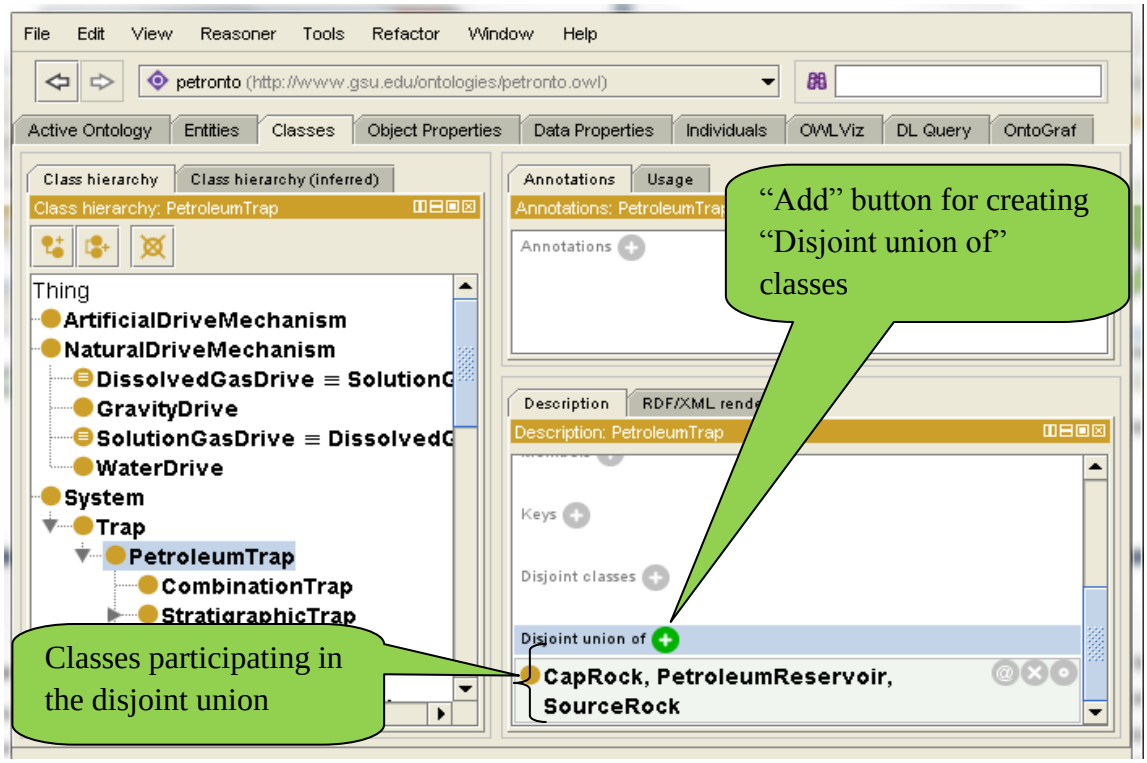

Figure 6.10. Screenshot showing the PetroleumTrap class being created as the disjoint union of CapRock, PetroleumReservoir, and SourceRock classes.

6.2.2 The Properties tab

The "Properties" tab is used to create and edit properties in the ontology. There are two major properties that may be created: object properties and data properties. Annotations are also a kind of property that may be created. Object properties relate the set of individuals of a class to the set of individuals of another class. The "Object Properties" tab is used for creating object properties. Figure 6.11 shows the "Object Properties" tab and relevant buttons for creating object properties.

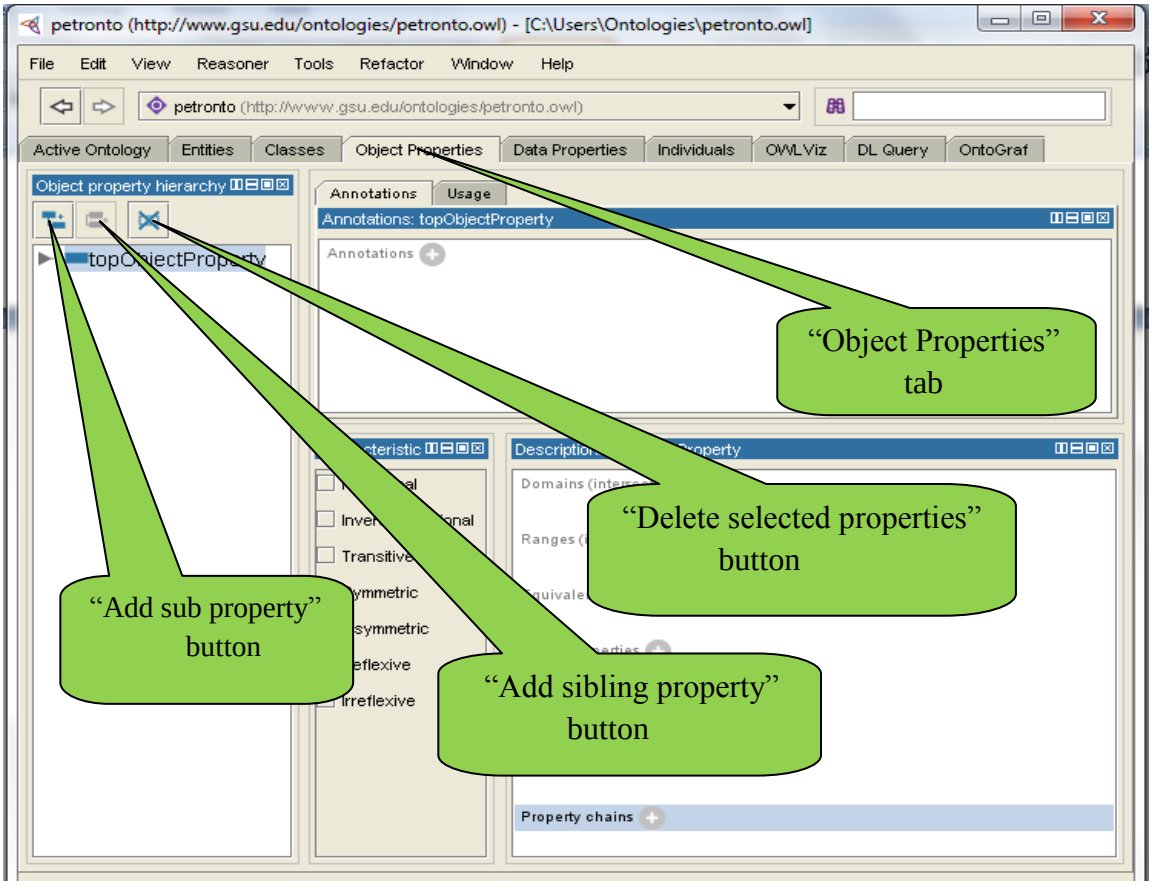

Figure 6.11. "Object Properties" tab and relevant buttons for creating object properties.

To begin creating object properties, we first click on the "Object Properties" tab, then on "topObjectProperty", of which every object property has to be a sub property. Next click on "Add sub property" button. A "Property Name Dialog" box then pops up into which we type or specify the name of the property to be created. Finally click on OK to create the property. Again, the convention adopted for creating property names, is to begin with lower case Arial font after which each word in the property's name begin with an upper case letter and are concatenated e.g., inContactWith. Figure 6.12 shows a pop up of the "Property Name Dialog" in which the property inContactWith is being created.

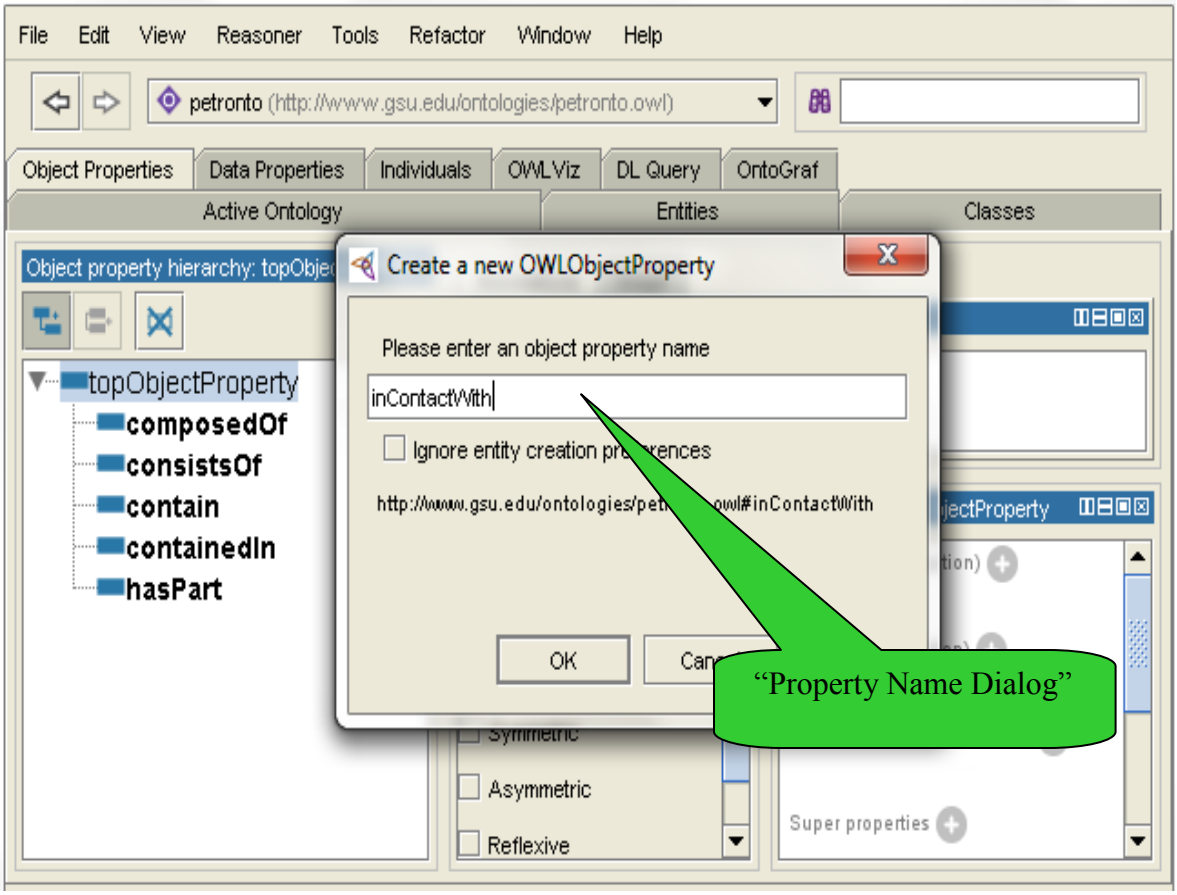

Figure 6.12. Shows pop up "Property Name Dialog" for creating ObjectProperties. In this figure, the property inContactWith is being created.

After creating an object property, its characteristic (Functional, Inverse functional, Transitive, Symmetric, etc) may then be specified by checking the appropriate box by the different property characteristics. Figure 6.13 shows the "Symmetric" characteristic assigned to the inContactWith property.

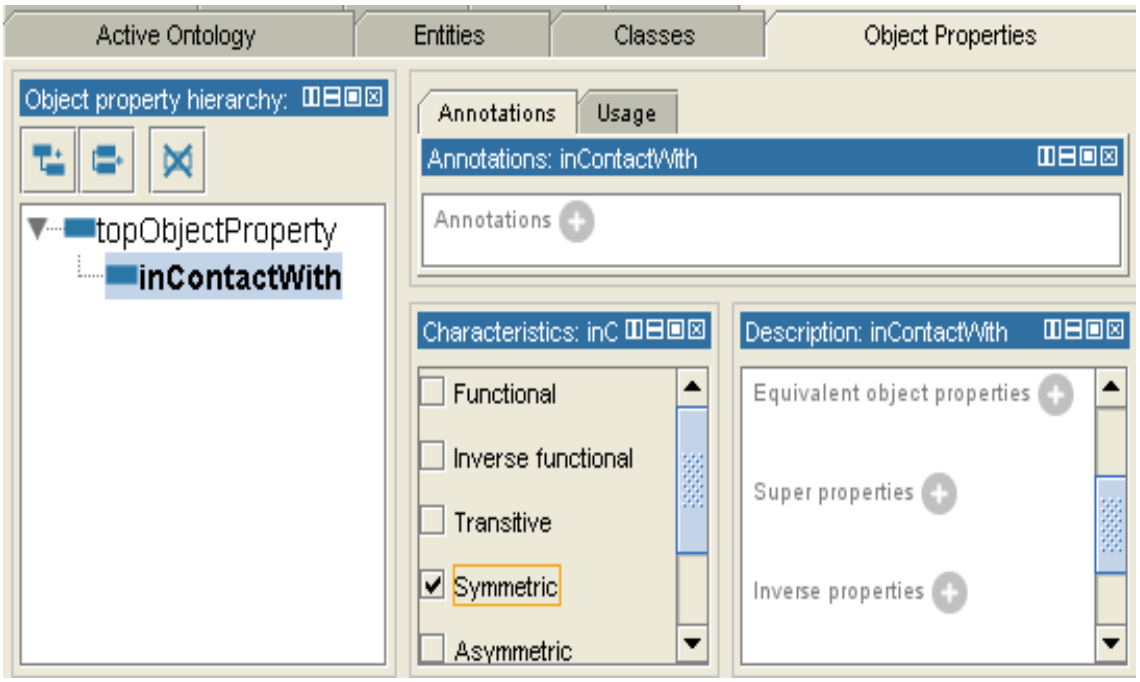

Figure 6.13. Assigning a property characteristic to a selected property. This figure shows the inContactWith property made "Symmetric".

The left section of the Properties tab (Object Properties tab or Data Properties tab) will show the property hierarchy. Details of any selected property are shown on the right section of the screen in a form. The properties show the domain, range, super properties, characteristics (Functional, Transitive etc.) and other features of the selected property.

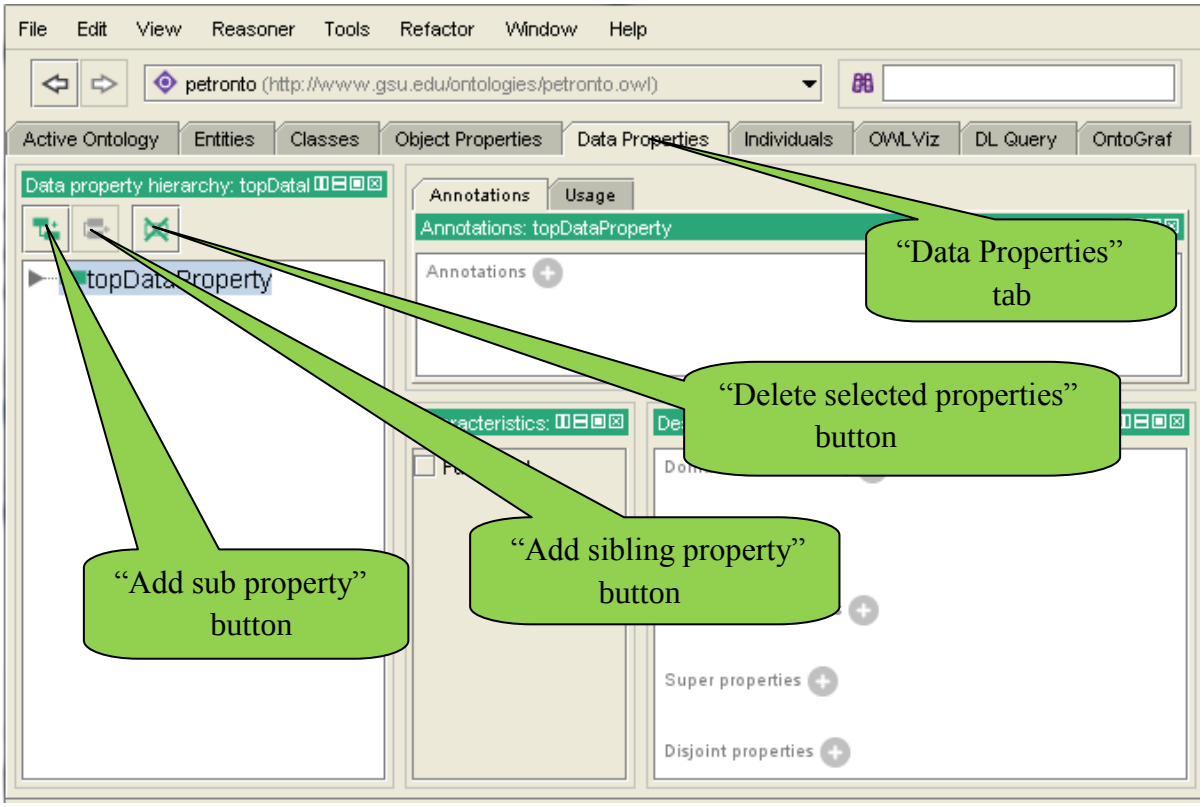

Figure 6.14. The "Data Properties" tab.

Datatype properties relate the set of individuals (instances) of a class to the set of individuals of a datatype. Datatype properties are created using the "Data Properties" tab Figure 6.14. To begin creating data properties such as hasSpecificGravity, first click on the "Data Properties" tab, then select "topDataProperty". Next, click on the "Add sub-property" button. A "Property Name Dialog" pops up, Figure 6.15. This figure shows the dialog into which is typed hasSpecificGravity. Finally press OK to complete the process. Notice again that a data type property may only have the characteristic of being functional.

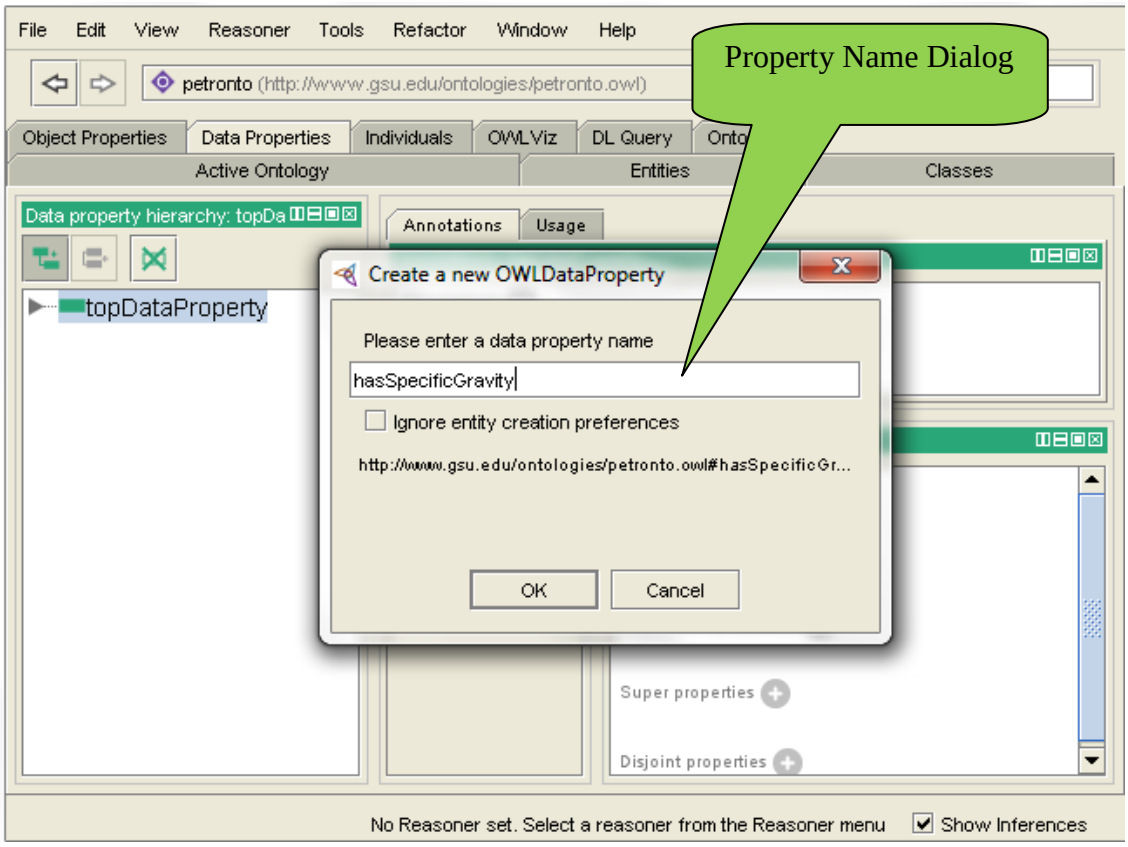

Figure 6.15. Screen shot of **"**Property Name Dialogue" for creating Data Properties. In this figure, the property, hasSpecificGravity is being created.

A sub property of any property is created by first selecting the property in the property hierarchy. Next, click on the "Add sub property" button. A "Property Name Dialog" box then pops up into which we type or specify the name of the property to be created, just as previously described. Figure 6.16 shows the creation of intergranularPorosity as a sub property of primaryPorosity.

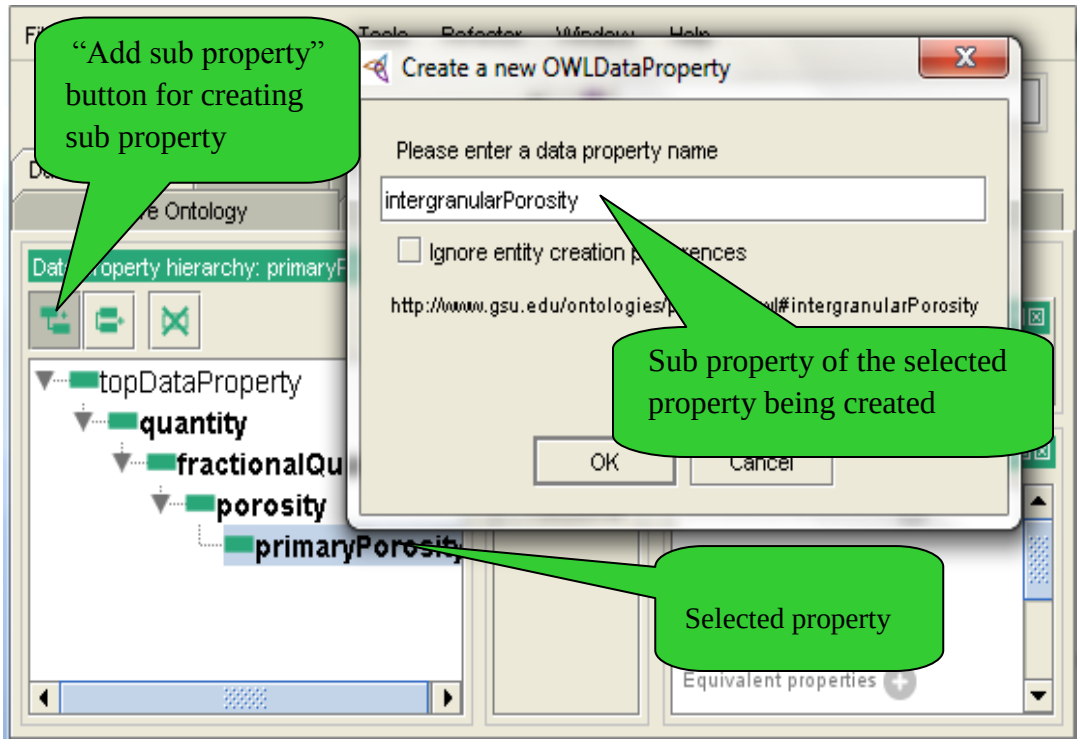

Figure 6.16. Screenshot showing the creation of intergranularPorosity as a sub-property of primaryPorosity.

When a property is defined, the domain is usually used to specify which class uses the property and the range is used to specify the values for the property. For example, the containedIn property may have Oil as its domain and ReservoirRock as its range so that the statement could be read: Oil containedIn ReservoirRock. To assign the domain of a property as in this example, first select the property (containedIn), from the property hierarchy, and then click on the "Add" button by the "Domains". The class hierarchy will open, through which we navigate and then select the Oil class. To assign the range, select the same property (containedIn), and then click on the "Add" button by the "Ranges". The class hierarchy will again open, from which we select the range (ReservoirRock) for the property. Figure 6.17 shows the domain and range of the containedIn property.

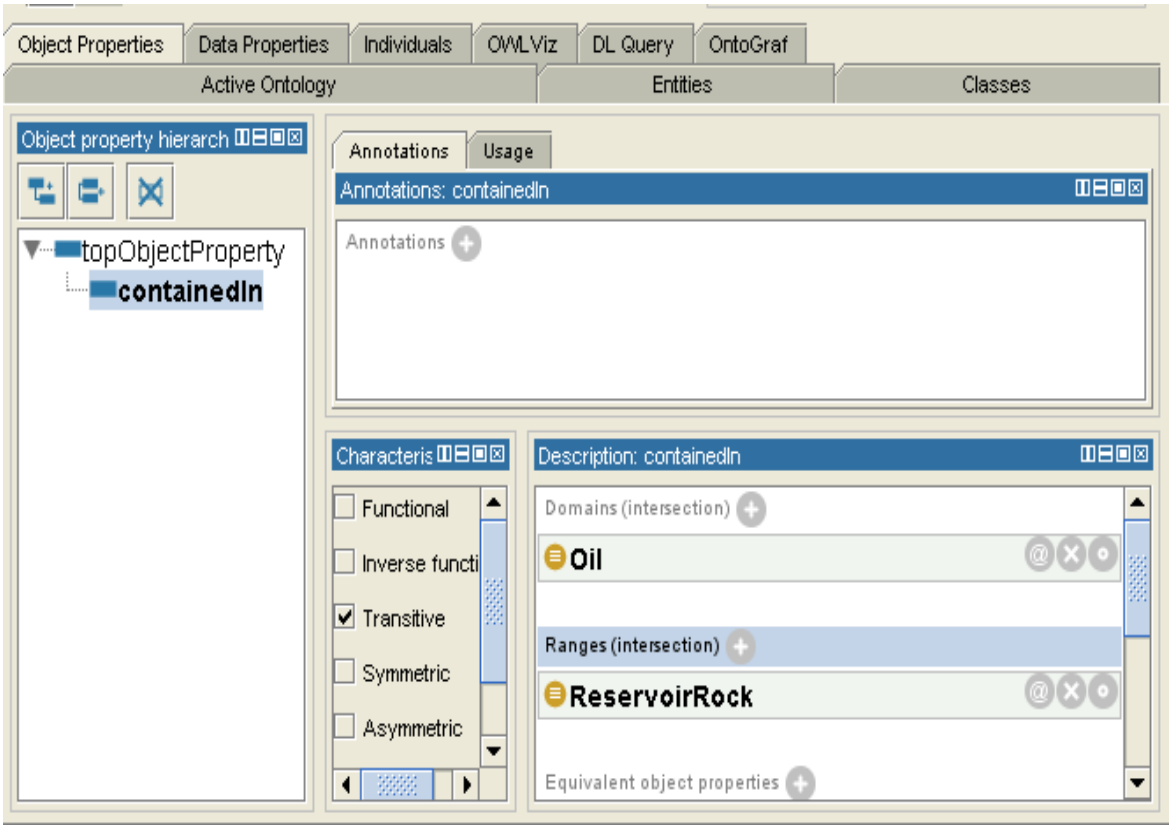

Figure 6.17. Screen shot showing the Domain and Range of the containedIn property.

6.2.4 The OWLViz tab

OWLViz is a visualization plugin for the Protégé OWL Plugin. The OWLViz plugin allows class hierarchies such as the asserted class hierarchy and the inferred class hierarchy in an OWL Ontology to be viewed, navigated and compared. The OWLViz tab also comes bundled with the full installation of Protégé and enabled through clicking on Window, Tabs, and on the checkbox by the OWLViz tab.

The OWLViz 4.1.1 version is compatible with the Protégé-OWL 4.1. Again, it is aided by Graphviz (http://www.graphviz.org), which is another visualization software from AT&T.

This means after installing Protégé, to use the OWLViz tab, the right version of the Graphviz visualition software needs to be installed as well. If Protégé is unable locate Graphviz, the path to the .exe file needs to be specified for the successful operation of OWLViz.

To display any class from the class hierarchy, the class is first selected and then the "Show class" button located on the OWLViz toolbar is clicked. A "class radius" dialogue box will then be shown. The class radius values indicate the number of levels of superclasses and subclasses that should be displayed around the selected class. For example, a class radius value of zero will show no superclass or subclass but only the selected class. A class radius of one will show one super, class and one subclass around the selected class and so forth.

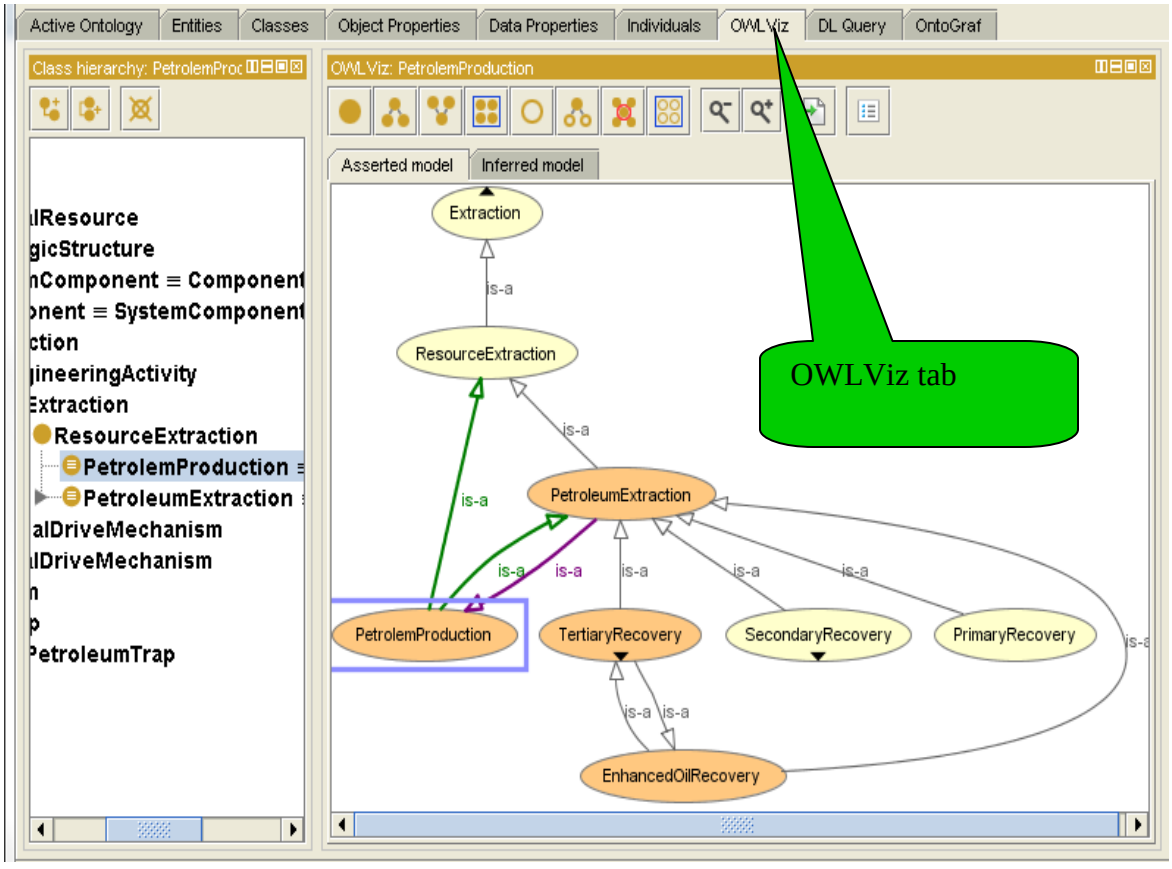

Figure 6.18. Screenshot of the OWLViz tab.

6.2.4 Constructing OWL Class Expressions

Earlier version of protégé makes use of Standard Description Logic (DL) (Baader et.al, 2003), symbols such as ¬, ≤, ≥, etc., in its logical expressions. The current Protégé 4.1 version used for this ontology however makes use of the Manchester OWL Syntax (http://www.co-ode.org/resources/reference/manchestersyntax). The Manchester OWL Syntax makes provision for the construction of OWL class expressions in a very lucid and user-friendly manner. The Manchester OWL Syntax replaces special symbols such as ∃, ∀, ⊓, ⊔, etc in DL Syntax with English language key words. For example, symbols such as ∃, ∀, and ⊔, are replaced with "some", "only", and "or", respectively. This makes expressions easier to understand. The meaning of these symbols is given in Table 6.1

Table 6.1. Various DL symbols and the Manchester OWL Syntax keywords.

OWL construct	**DL symbol**	**Manchester OWL syntax keyword**	**Example**
someValuesFrom	∃	**some**	hasContent **some** HydroCarbon
allValuesFrom	∀	**only**	hasRockType **only** Sedimentary
hasValue	∋	**value**	hasLocation **value** Texas
minCardinality	≥	**min**	hasChemicalElementType **min** 2
Cardinality	=	**exactly**	hasDrilledOilWell **exactly** 8
maxCardinality	≤	**max**	hasProductionWell **max** 5
intersectionOf	⊓	**and**	HydroCarbon **and** Liquid
unionOf	⊔	**or**	ReservoirRock **or** CapRock
complementOf	¬	**not**	**not** Igneous

6.2.5 Description Logic (DL) Reasoning

Description logic (DL) reasoning is done on OWL DL ontologies using reasoners such as Fact+ +, HermiT, or Pellet.When invoked, the reasoner performs two major operations or tests: consistency checking and classification.

Consistency checking is the test whether it is appropriate for a class to have instances or not. This is done by the reasoner on the bases of the class definitions. Classes identified to be inconsistent cannot have instances i.e., it will be an error assigning instances to inconsistent classes. For example, if the Oil and NaturalGas classes are defined to be disjoint, a third class HighEfficiencyFuel cannot be defined to be a subclass of both Oil and NaturalGas. Upon classification, the reasoner will quickly detect that HighEfficiencyFuel is an inconsistent class and should not have an instance. This is because HighEfficiencyFuel class has been defined as a subclass of two disjoint classes. If it should have instances, then its instances should also be instances of the two disjoint classes, which cannot be the case. Such inconsistent classes are highlighted in red after the classification is completed.

Classification is the test or check to find out if a class is a subclass of another class (also known as subsumption). It is invoked with the "Classify" button. The reasoner makes use of the descriptions of classes to further establish any subclass or superclass relationship between classes. After the reasoner has completed the classification process, another hierarchy known as inferred hierarchy (computed class hierarchy by the reasoner) is displayed alongside the asserted hierarchy (manually constructed class hierarchy). Classes whose positions have changed (i.e. classes that changed their superclasses) after classification, will appear on a blue background in the inferred hierarchy.

CHAPTER 7

RESULTS

Extensions made to the SWEET ontologies are discussed in this Chapter. The concepts mentioned in this Chapter are defined in detail in Chapter 2.

7.1 Extensions made to *matrNaturalResource.owl*

The *matrNaturalResource.owl* file as the name implies contains concepts related to natural resources such as Petroleum, CrudeOil, NaturalGas and Coal.

7.1.1 The concept of Petroleum redefined and remodeled

In the *matrNaturalResource.owl* ontology, Petroleum has been essentially defined as a liquid consisting of naturally occurring hydrocarbons. Petroleum is redefined to include both liquid and gaseous naturally occurring hydrocarbons, and modeled accordingly. This definition is given in Chapter 2. Thus Petroleum includes both crude oil and natural gas. In the original modeling, the *matrNaturalResource.owl* file has the class NaturalResource and a subclass FossilFuel. FossilFuel further has Oil as a subclass. The class Petroleum was modeled as subclass of the Oil class. This conceptual error is corrected and the Oil class is instead made a subclass of the Petroleum class. Again the class NaturalGas, which was originally directly placed under FossilFuel as a subclass is also re-modeled and made a subclass of the Petroleum class. The *matrOrganicCompound.owl* file has the concept Hydrocarbon. Using the composedOf property, the owl:someValuesFrom construct, and the Hydrocarbon class, Petroleum is modeled to be composed of at least (some) hydrocarbons. Crude and CrudeOil are also made equivalent to Oil.

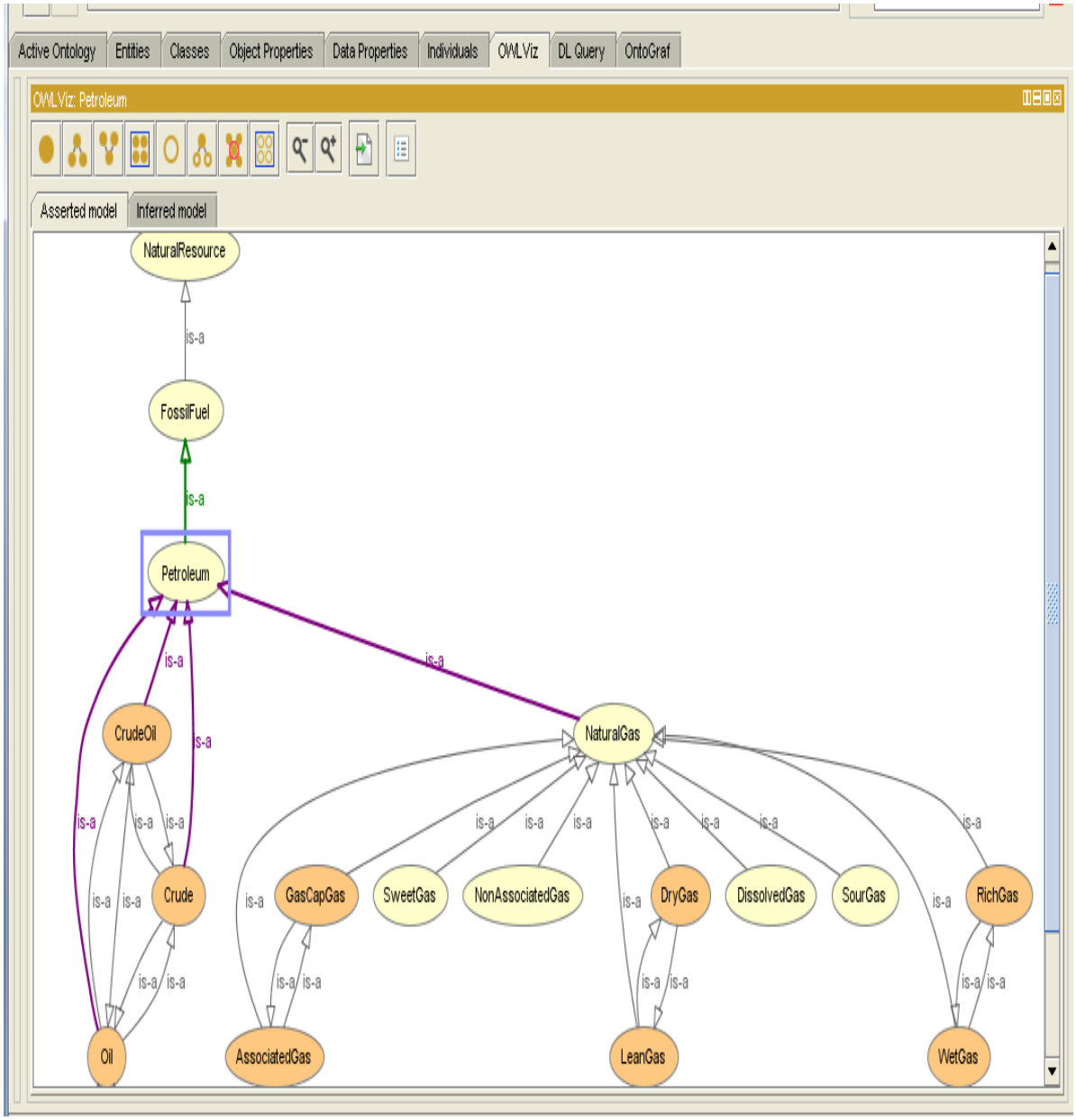

Figure 7.1. Screen shot showing subclasses and superclasses of Petroleum

7.1.2 Multiple more specific concepts added to NaturalGas

Hydrocarbon natural gas is a gas composed of a mixture of hydrocarbon molecules that have one, two, three or four, carbon atoms. It is subclassed into DryGas (mainly methane) and WetGas (ethane, propane, and butane). Dry gas is a hydrocarbon natural gas containing no liquid condensate or minor amount of liquid condensate. It is also referred to as lean gas. Hence the class DryGas is made equivalent to LeanGas. Wet gas also referred to as rich gas contains significant amounts of condensate. The WetGas class is made equivalent to RichGas. NaturalGas is also subclassed into DissolvedGas, AssociatedGas and NonAssociatedGas. Dissolved gas is natural gas in solution in crude oil, which is contained in the reservoir. Associated gas is natural gas that overlies and is in contact with crude oil in the reservoir. AssociatedGas is further made equivalent to the GasCapGas class. Nonassociated gas is natural gas that is not in contact with crude oil in the reservoir (such reservoirs do not contain significant quantities of crude oil). The NaturalGas class is also subclassed into SweetNaturalGas and SourNaturalGas. Sweet natural gas (or sweetgas) is natural gas containing very little or no sulphur or sulphur compounds. Sour natural gas (or sour gas) is natural gas containing significant quantities of sulphur or sulphur compounds.

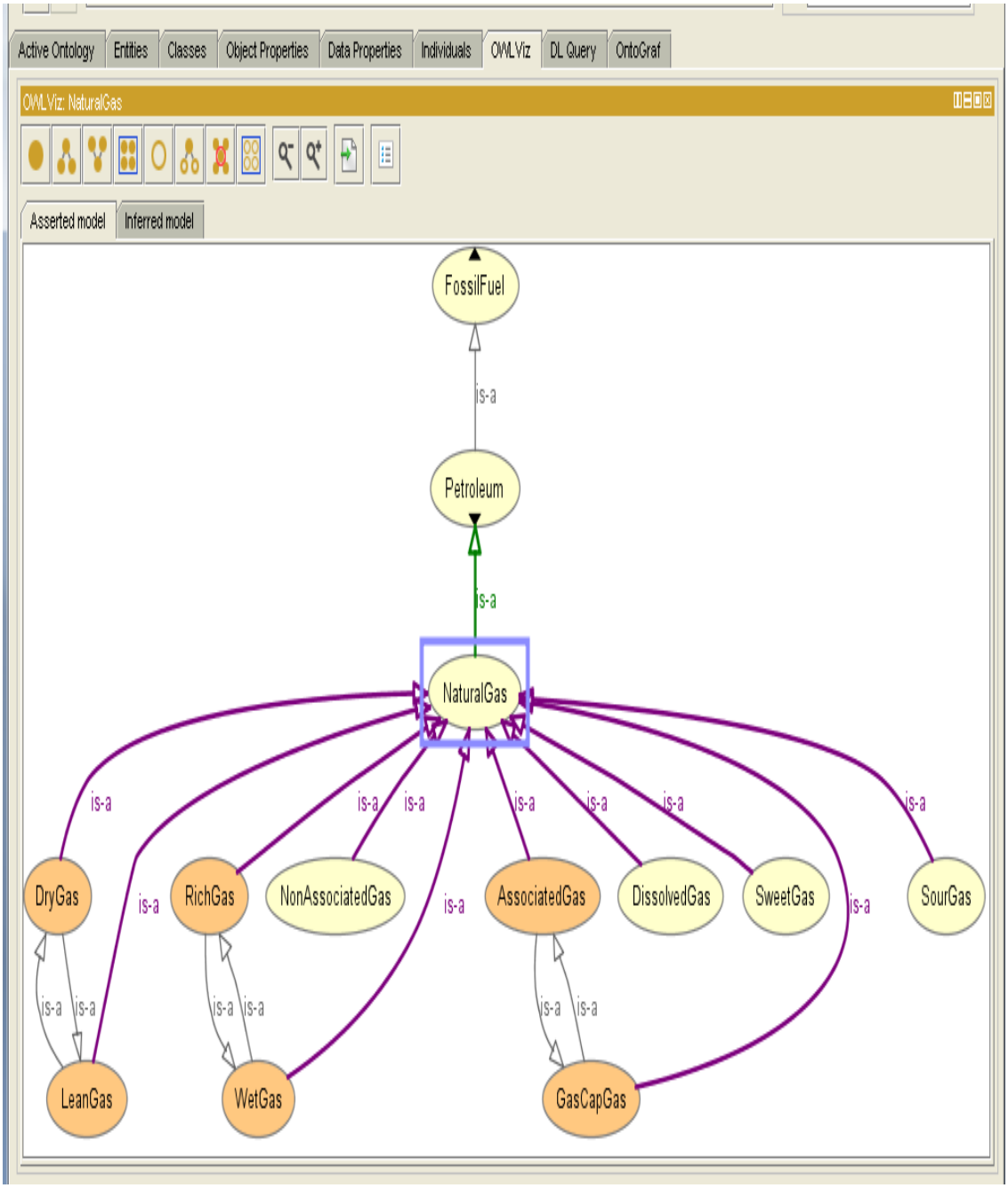

Figure 7.2. Screen shot showing subclasses and superclasses of NaturalGas

Relevant Codes:

List 7.1 Code showing the a restriction on the composedOf object property to assert that Petroleum is composed of atleast (some) Hydrocarbons. This construct means that, in addition to hydrocarbons, petroleum may be composed of other elements.

```
<!-- http://www.gsu.edu/ontologies/petronto.owl#Petroleum -->


    <owl:Class rdf:about="&petronto;Petroleum">
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="&petronto;composedOf"/>
                <owl:someValuesFrom rdf:resource="&petronto;Hydrocarbons"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>
```

List 7.2 Code showing the use of the containedIn object property to assert that Petroleum is contained in PetroleumReservoir. Petroleum is the domain of the property, and PetroleumReservoir is the range of the property. The code also shows that contain and containedIn are inverse properties, hence PetroleumReservoir contain Petroleum.

```
<!-- http://www.gsu.edu/ontologies/petronto.owl/containedIn -->

    <owl:ObjectProperty rdf:about="&petronto;containedIn">
        <rdfs:domain rdf:resource="&petronto;Petroleum"/>
        <rdfs:range rdf:resource="&petronto;PetroleumReservoir"/>
        <owl:inverseOf rdf:resource="&petronto;contain"/>
    </owl:ObjectProperty>
```

List 7.3 This code shows that the AssociatedGas class is equivalent to the GasCapGas class. The code further indicates that AssociatedGas is a subclass of NaturalGas.

```
<!-- http://www.gsu.edu/ontologies/petronto.owl/AssociatedGas -->

    <owl:Class rdf:about="&petronto;AssociatedGas">
        <owl:equivalentClass rdf:resource="&petronto;GasCapGas"/>
        <rdfs:subClassOf rdf:resource="&petronto;NaturalGas"/>
    </owl:Class>
```

List 7.4 Code about the Crude class. This code indicates that the Crude class is equivalent to the CrudeOil class, and also to the Oil class. The code further indicates that Crude is a subclass of the Petroleum class.

```
<!-- http://www.gsu.edu/ontologies/petronto.owl/Crude -->

   <owl:Class rdf:about="&petronto;Crude">
      <owl:equivalentClass rdf:resource="&petronto;CrudeOil"/>
      <owl:equivalentClass rdf:resource="&petronto;Oil"/>
      <rdfs:subClassOf rdf:resource="&petronto;Petroleum"/>
   </owl:Class>
```

List 7.5 Code about the DryGas class. The code shows that the DryGas class is equivalent to the LeanGas class, meaning they are the same classes. The code also shows that DryGas is a subclass of NaturalGas. Finally, the code indicates that DryGas is disjoint with WetGas.

```
 <!-- http://www.gsu.edu/ontologies/petronto.owl/DryGas -->

   <owl:Class rdf:about="&petronto;DryGas">
      <owl:equivalentClass rdf:resource="&petronto;LeanGas"/>
      <rdfs:subClassOf rdf:resource="&petronto;NaturalGas"/>
      <owl:disjointWith rdf:resource="&petronto;WetGas"/>
   </owl:Class>
```

List 7.6 Code about the SourGas class. The code shows that the SourGas class is a subclass of the NaturalGas class. The code further indicates that SourGas is disjoint with SweetGas.

```
<!-- http://www.gsu.edu/ontologies/petronto.owl/SourGas -->

    <owl:Class rdf:about="&petronto;SourGas">
        <rdfs:subClassOf rdf:resource="&petronto;NaturalGas"/>
        <owl:disjointWith rdf:resource="&petronto;SweetGas"/>
    </owl:Class>
```

List 7.7 Code describing the AssociatedGas, DissolvedGas, and the NonAssociatedGas classes as all pair-wise disjoint classes.

```
 <rdf:Description>
        <rdf:type rdf:resource="&owl;AllDisjointClasses"/>
        <owl:members rdf:parseType="Collection">
            <rdf:Description rdf:about="&petronto;AssociatedGas"/>
            <rdf:Description rdf:about="&petronto;DissolvedGas"/>
            <rdf:Description rdf:about="&petronto;NonAssociatedGas"/>
        </owl:members>
    </rdf:Description>
```

7.2 Extensions made to *propFraction.owl*

The *propFraction.owl* file consists of concepts related to fractional properties such as Humidity, MoleFraction, and Porosity.

7.2.1 Multiple concepts added to Porosity

The *propFraction.owl* file has the class FractionalQuantity, which has the subclass Porosity.

The following new classes are now added as subclasses of Porosity. PrimaryPorosity and SecondaryPorosity are made subclasses of Porosity. PrimaryPorosity is again made equivalent to DepositionalPorosity. SecondaryPorosity is also made equivalent to PostDepositionalPorosity. The various porosities are discussed in detail in Chapter 2.

PrimaryPorosity is further subclassed (divided) into IntergranularPorosity and IntragranularPorosity. Intergranular porosity occurs between grains. Intragranular porosity exists within grains, especially within grains of carbonate sands. IntergranularPorosity is made equivalent to InterparticlePorosity, while IntragranularPorosity is also made equivalent to IntraparticlePorosity. SecondaryPorosity is also further divided into the following subclasses: IntercrystalinePorosity, FenestralPorosity, VuggyPorosity, CavenousPorosity, MoldicPorosity, and FracturePorosity. Moldic porosity is the result of the selective dissolution of minerals, commonly carbonates, in which only the grains or only the matrix has been leached out. Vuggy porosity which is also the result of the dissolution of minerals, cuts across grains and matrix, and is often larger than moldic pores. Larger forms, vuggy porosity becomes known as cavenous porosity.

7.3 Extensions made to *matrRock.owl* and *reprSciComponent.owl*

The *matrRock.owl* file consists of concepts such as, Rock, RockBody, MetamorphicRock and SedimentaryRock. The *reprSciComponent.owl* consists of concepts including Reservoir.

7.3.1 Multiple concepts added to SedimentaryRock

The *matrRock.owl* file includes the Rock class, which also has the subclass SedimentaryRock. Multiple subclasses are added to SedimentaryRock such as SiliciclasticSedimentaryRock, ChemicalSedimentaryRock and BiochemicalSedimentaryRock. Some specialized types of these rocks which commonly form petroleum reservoir rocks, such as Sandstone and Limestone are also added.

7.3.2 PetroleumReservoir class and its subclasses

The *reprSciComponent.owl* file has the concept Reservoir. This Reservoir is not linked to a rock. Therefore a new class, PetroleumReservoir is created from the intersection of the RockUnit and Reservoir classes. This is done to assert that a PetroleumReservoir is a rock which stores or contains petroleum. The PetroleumReservoir class is further specified to contain at least (some) petroleum. This is modelled using the contain property and the owl:someValuesFrom construct with value from the Petroleum class. The kinds of PetroleumReservoir modeled are SandstoneReservoir, CarbonateReservoir and FracturedReservoir. SandstoneReservoir is created from the intersection of the PetroleumReservoir class and the Sandstone class. CarbonateReservoir is created from the intersection of the CarbonateRock class and the PetroleumReservoir class.

The following classes are made subclasses of SandstoneReservoir: DuneSandstoneReservoir, ShorelineSandstoneReservoir, RiverSandstoneReservoir and DeltaSandstoneReservoir. CarbonateReservoir is specialized into LimestoneReservoir and DolostoneReservoir. DolostoneReservoir is made an equivalent class of DolomiteReservoir. LimestoneReservoir is further specialized as ReefReservoir, CarbonateSandReservoir, CarbonateChalkReservoir. CarbonateChalkReservoir is made the equivalent class of CarbonateMudReservoir.

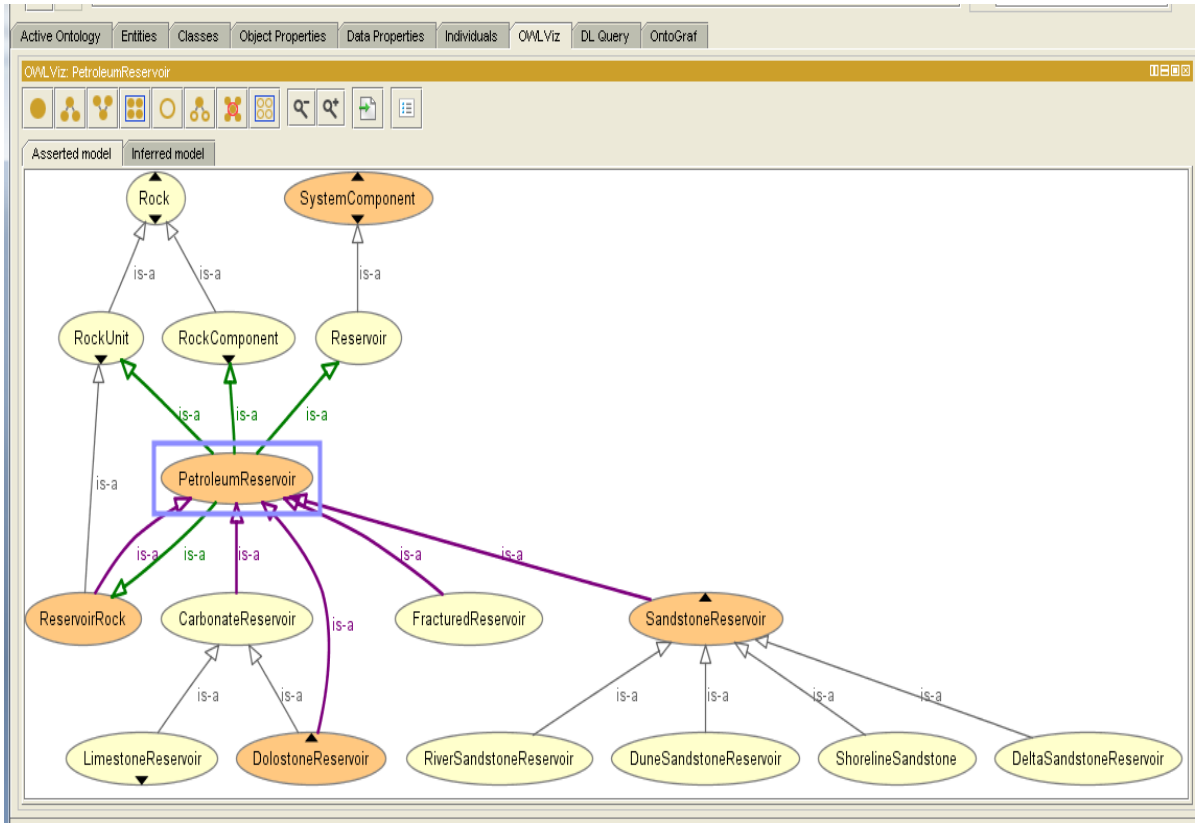

Figure 7.3. Screen shot showing subclasses and superclasses of the PetroleumReservoir class

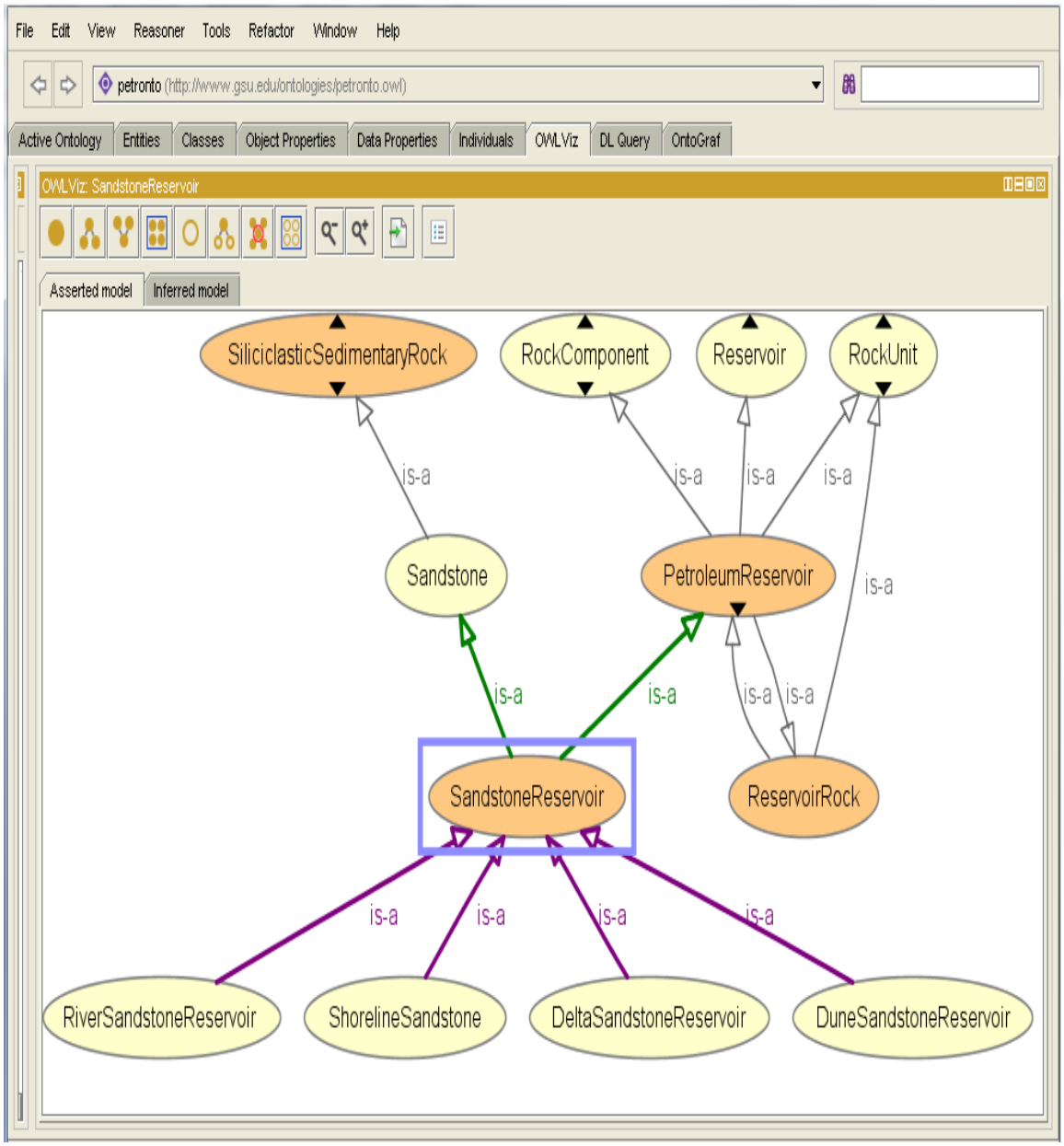

Figure 7.4. Screen shot showing the hierarchy of the SandstoneReservoir, and its subclasses and superclasses

List 7.8 Code showing that SandstoneReservoir, CarbonateReservoir, and FracturedReservoir are all disjoint classes. We are considering fractutred reservoirs to be other reservoirs which are not sandstone or carbonate reservoirs. This include fractured shale.

```
<rdf:Description>
    <rdf:type rdf:resource="&owl;AllDisjointClasses"/>
    <owl:members rdf:parseType="Collection">
        <rdf:Description rdf:about="&petronto;CarbonateReservoir"/>
        <rdf:Description rdf:about="&petronto;FracturedReservoir"/>
        <rdf:Description rdf:about="&petronto;SandstoneReservoir"/>
    </owl:members>
</rdf:Description>
```

List 7.9 Code using the inContactWith object property to assert that PetroleumReservoir is in contact with the CapRock as well as the SourceRock. In other words, two ranges are defined for the inContactWith property whose domain is PetroleumReservoir.

```
<-- http://www.gsu.edu/ontologies/petronto.owl/inContactWith -->

<owl:ObjectProperty rdf:about="&petronto;inContactWith">
      <rdf:type rdf:resource="&owl;SymmetricProperty"/>
       <rdfs:domain rdf:resource="&petronto;PetroleumReservoir"/>
      <rdfs:range rdf:resource="&petronto;CapRock"/>
      <rdfs:range rdf:resource="&petronto;SourceRock"/>
</owl:ObjectProperty>
```

List 7.10 Code showing the construction of the PetroleumReservoir class from the intersection of the Reservoir class and the RockUnit class. The code again shows that PetroleumReservoir is made equivalent to ReservoirRock.

```
<!-- http://www.gsu.edu/ontologies/petronto.owl/PetroleumReservoir -->


  <owl:Class rdf:about="&petronto;PetroleumReservoir">
    <owl:equivalentClass rdf:resource="&petronto;ReservoirRock"/>
     <owl:equivalentClass>
       <owl:Class>
          <owl:intersectionOf rdf:parseType="Collection">
             <rdf:Description rdf:about="&petronto;Reservoir"/>
             <rdf:Description rdf:about="&petronto;RockUnit"/>
          </owl:intersectionOf>
       </owl:Class>
     </owl:equivalentClass>
  </owl:Class>
```

List 7.11 Code showing the construction of a CarbonateReservoir from the intersection of PetroleumReservoir class and the CarbonateRock class. The code again shows that CarbonateReservoir is a subclass of PetroleumReservoir.

```
<!-- http://www.gsu.edu/ontologies/petronto.owl/CarbonateReservoir -->


<owl:Class rdf:about="&petronto;CarbonateReservoir">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="&petronto;PetroleumReservoir"/>
        <rdf:Description rdf:about="&petronto;CarbonateRock"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="&petronto;PetroleumReservoir"/>
</owl:Class>
```

List 7.12 Code showing the construction of a DolostoneReservoir from the intersection of CarbonateReservoir class and the Dolostone class. The code again shows that DolostoneReservoir is a subclass of CarbonateReservoir.

```
<!-- http://www.gsu.edu/ontologies/petronto.owl/DolostoneReservoir -->


<owl:Class rdf:about="&petronto;DolostoneReservoir">
   <owl:equivalentClass>
      <owl:Class>
         <owl:intersectionOf rdf:parseType="Collection">
            <rdf:Description rdf:about="&petronto;Dolostone"/>
            <rdf:Description rdf:about="&petronto;CarbonateReservoir"/>
         </owl:intersectionOf>
      </owl:Class>
   </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="&petronto;CarbonateReservoir"/>
</owl:Class>
```

List 7.13 Code showing the construction of a LimestoneReservoir from the intersection of CarbonateReservoir class and the Limestone class. The code again shows that LimestoneReservoir is a subclass of CarbonateReservoir.

```
<!-- http://www.gsu.edu/ontologies/petronto.owl/LimestoneReservoir -->

<owl:Class rdf:about="&petronto;LimestoneReservoir">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="&petronto;Limestone"/>
        <rdf:Description rdf:about="&petronto;CarbonateReservoir"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="&petronto;CarbonateReservoir"/>
</owl:Class>
```

List 7.14 Code showing the construction of a SandstoneReservoir from the intersection of the PetroleumReservoir class and the Sandstone class. The code again shows that SandstoneReservoir is a subclass of PetroleumReservoir.

```
<!-- http://www.gsu.edu/ontologies/petronto.owl/SandstoneReservoir -->


<owl:Class rdf:about="&petronto;SandstoneReservoir">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="&petronto;PetroleumReservoir"/>
        <rdf:Description rdf:about="&petronto;Sandstone"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="&petronto;PetroleumReservoir"/>
</owl:Class>
```

7.4 Extensions made to *reprSciComponent.owl* to create PetroleumTrap

The *reprSciComponent.owl* file consists of concepts such as Component, SystemComponent etc.

7.4.1 Multiple concepts added to PetroleumTrap.

A new class, System is introduced which is connected to SystemComponent using the hasPart property. Trap is also created as a subclass of System, and a new class PetroleumTrap is made a subclass of Trap. PetroleumTrapComponent is made a subclass of SystemComponent. PetroleumTrapComponent is constructed to have as its parts RockComponent and StructuralComponent using the consistsOf property. RockComponent is specialized into SourceRock, ReservoirRock, and CapRock. StructuralComponent is specialized into Fold and Contact. Further subclasses of Contact are created as Fault, Intrusion, Depositional, and Unconformity.

PetroleumTrap is essentially modeled as the owl:disjointUnionOf the SourceRock, ReservoirRock, and CapRock. The classes StructuralTrap, StratigrapicTrap and CombinationTrap are created as subclasses of PetroleumTrap.

AnticlinalTrap, FaultTrap and DiarpiricTrap are created as subclasses of StructuralTrap. AnticlinalTrap is also made an equivalent class of FoldTrap. DiarpiricTrap is further specialized as SaltDomeTrap and MudDiarpirTrap. SaltDomeTrap is the same as a SaltPlugTrap. These classes are therefore made equivalent classes. StratigraphicTrap is specialized into UnconformityTrap, DiageneticTrap and DepositionalTrap.

The above classes may be further split into more specialized classes (Selley, 1998) as shown below:

AnticlinalTrap may further have CompressionalAnticlinalTrap (caused by crustal shortening) and CompactionalAnticlinalTrap (caused by crustal extension) as subclasses. The FaultTrap class may also be further separated into TransverseFaultTrap and TensionalFaultTrap.

SupraUnconformityTrap and SubUnconformityTrap may be made subclasses of UnconformityTrap. DepositionalTrap is specialized as ChannelTrap, BarrierBarTrap, PinchoutTrap, and CarbonateReefTrap.

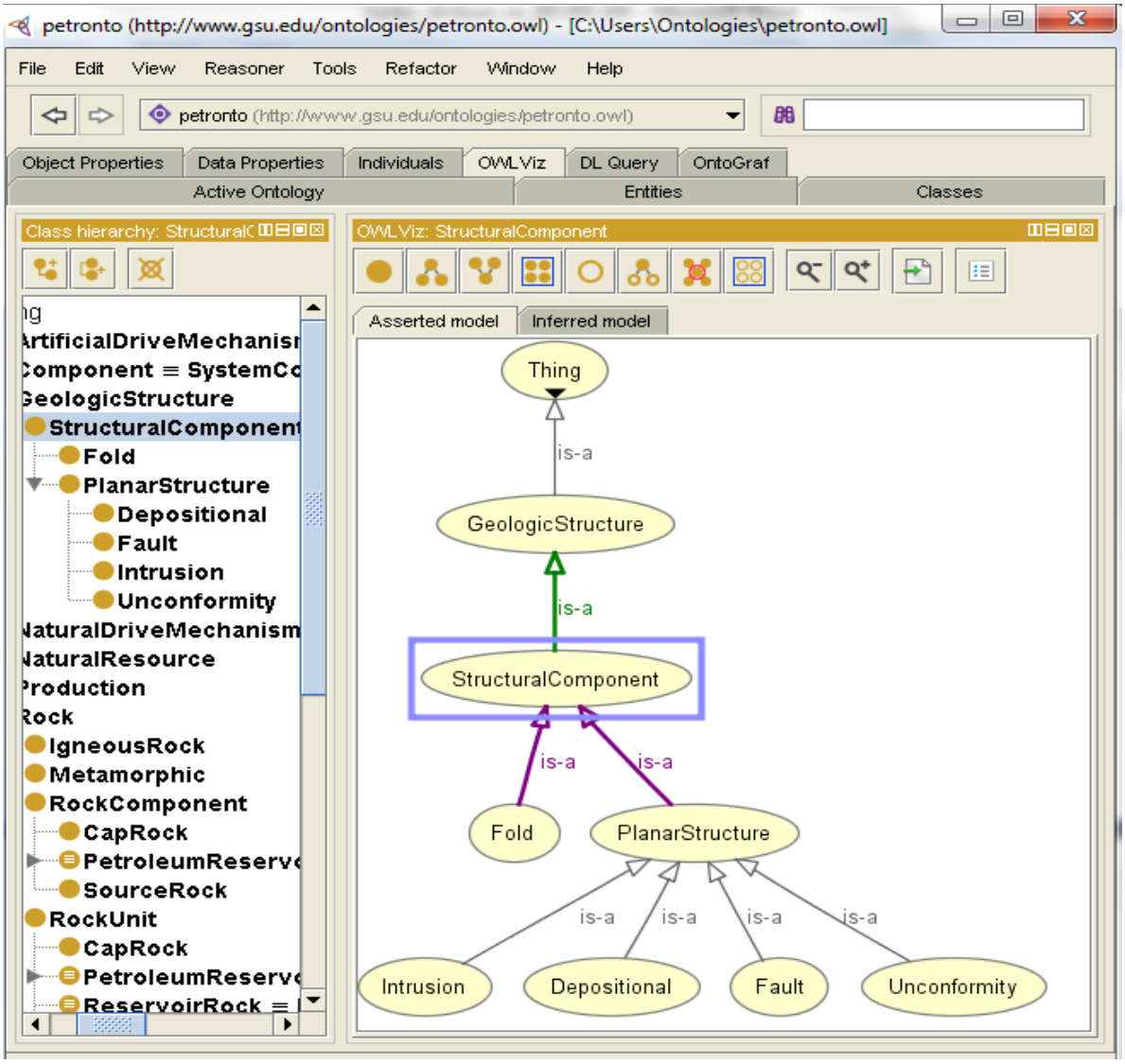

Figure 7.5. Screen shot showing the StructuralComponent of a PetroleumTrap

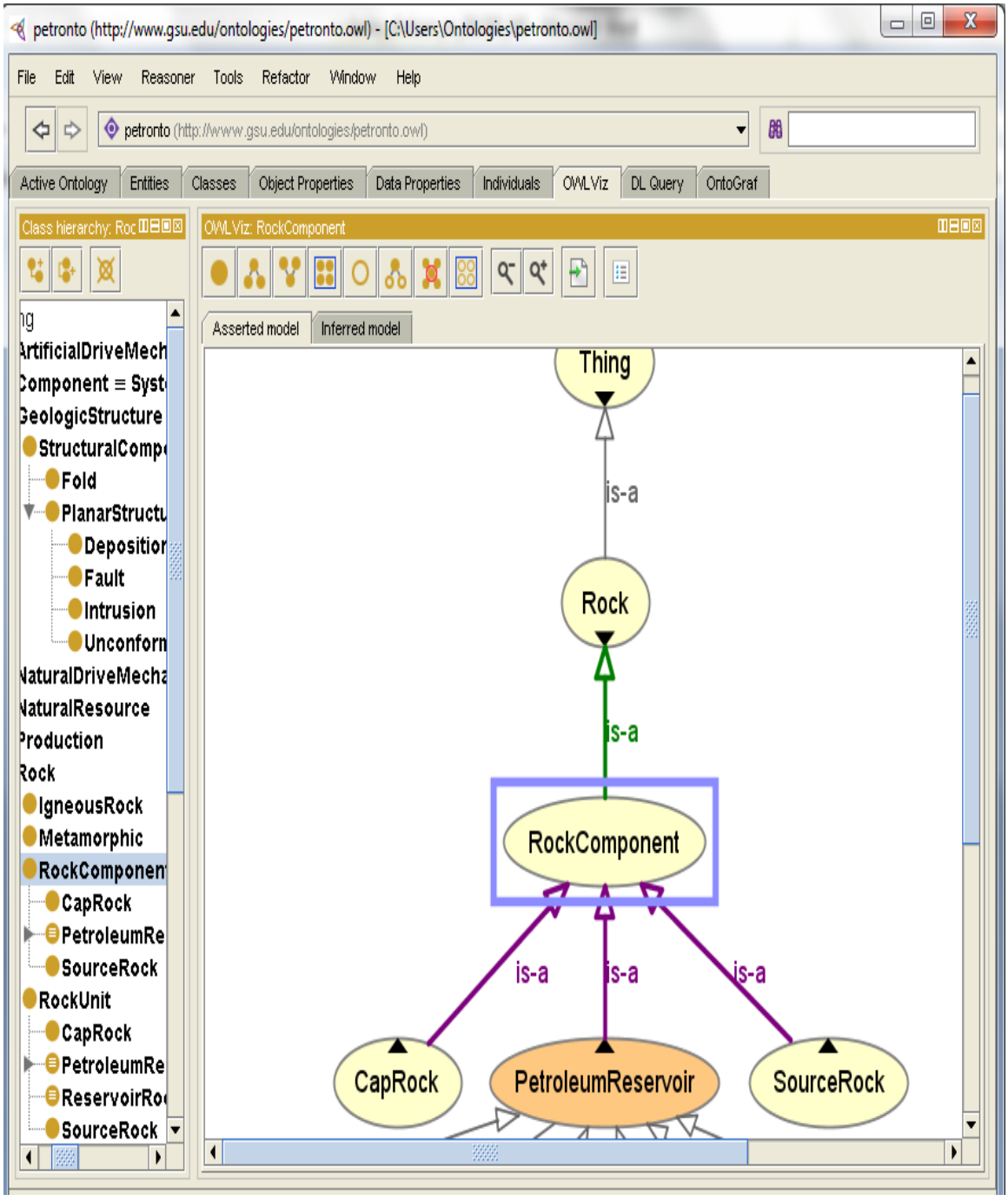

Figure 7.6. Screen shot showing the RockComponent of PetroleumTrap

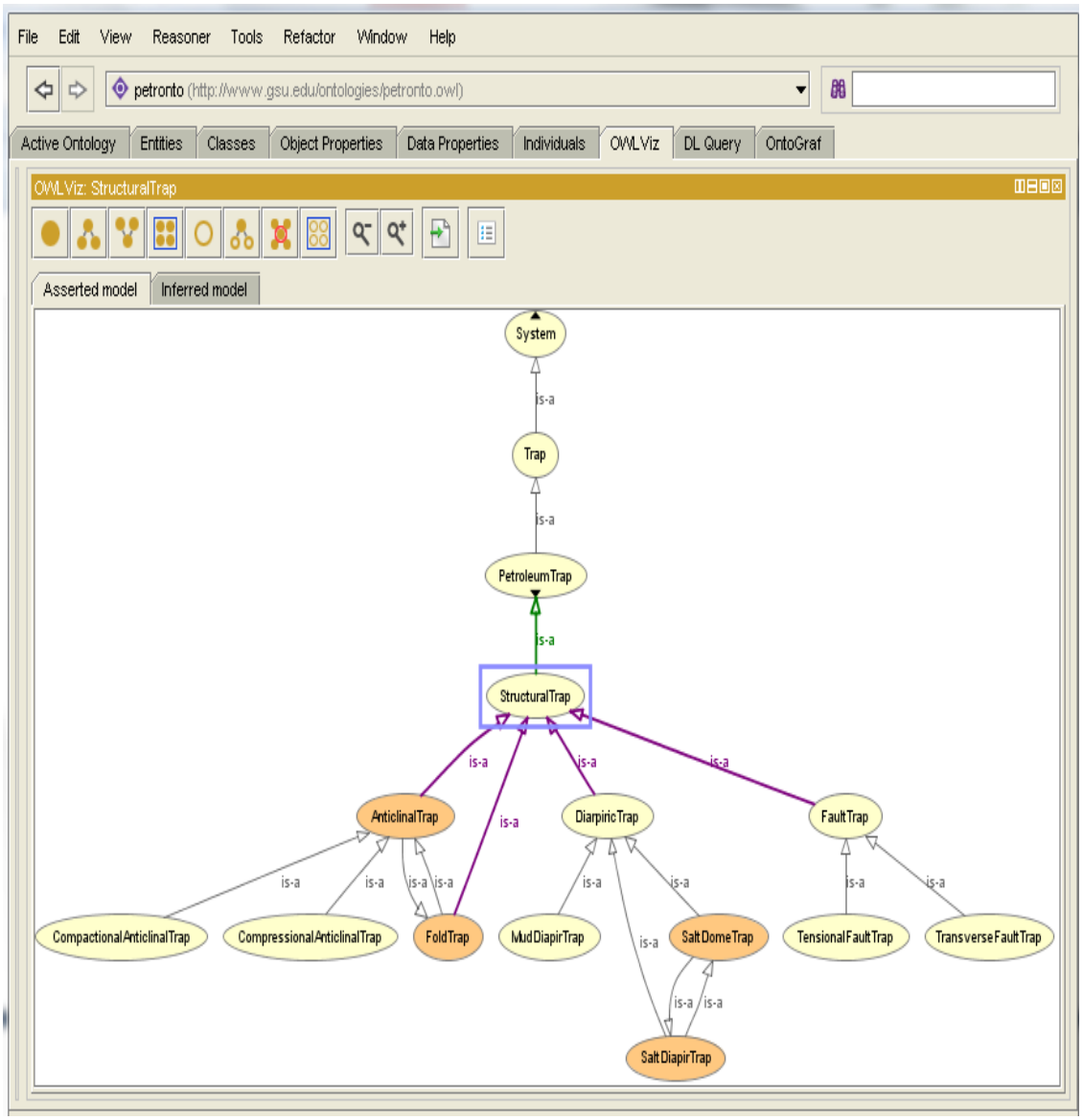

Figure 7.7. Screen shot showing specializations of the StructuralTrap

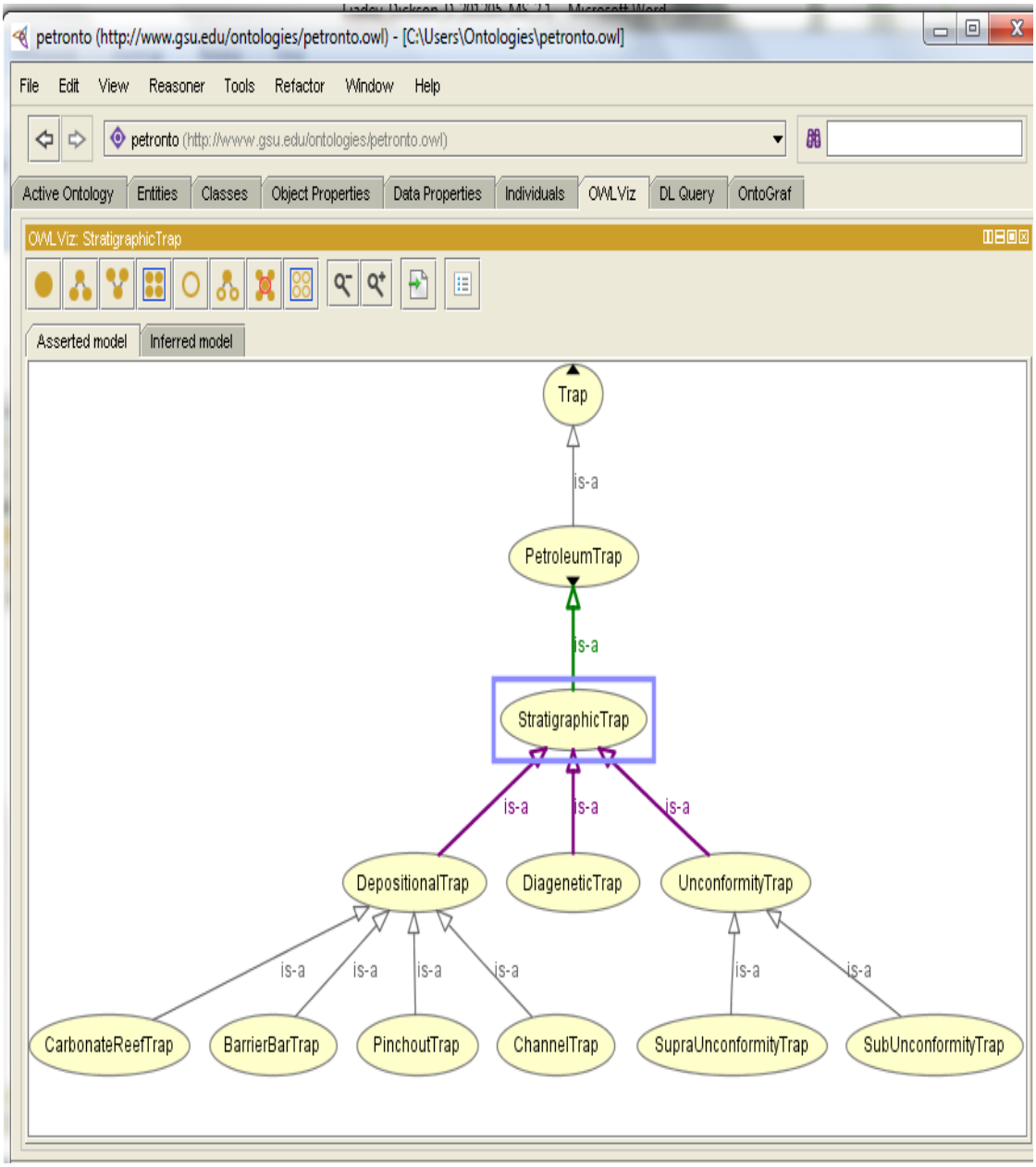

Figure 7.8. Screen shot showing specializations of the StratigraphicTrap

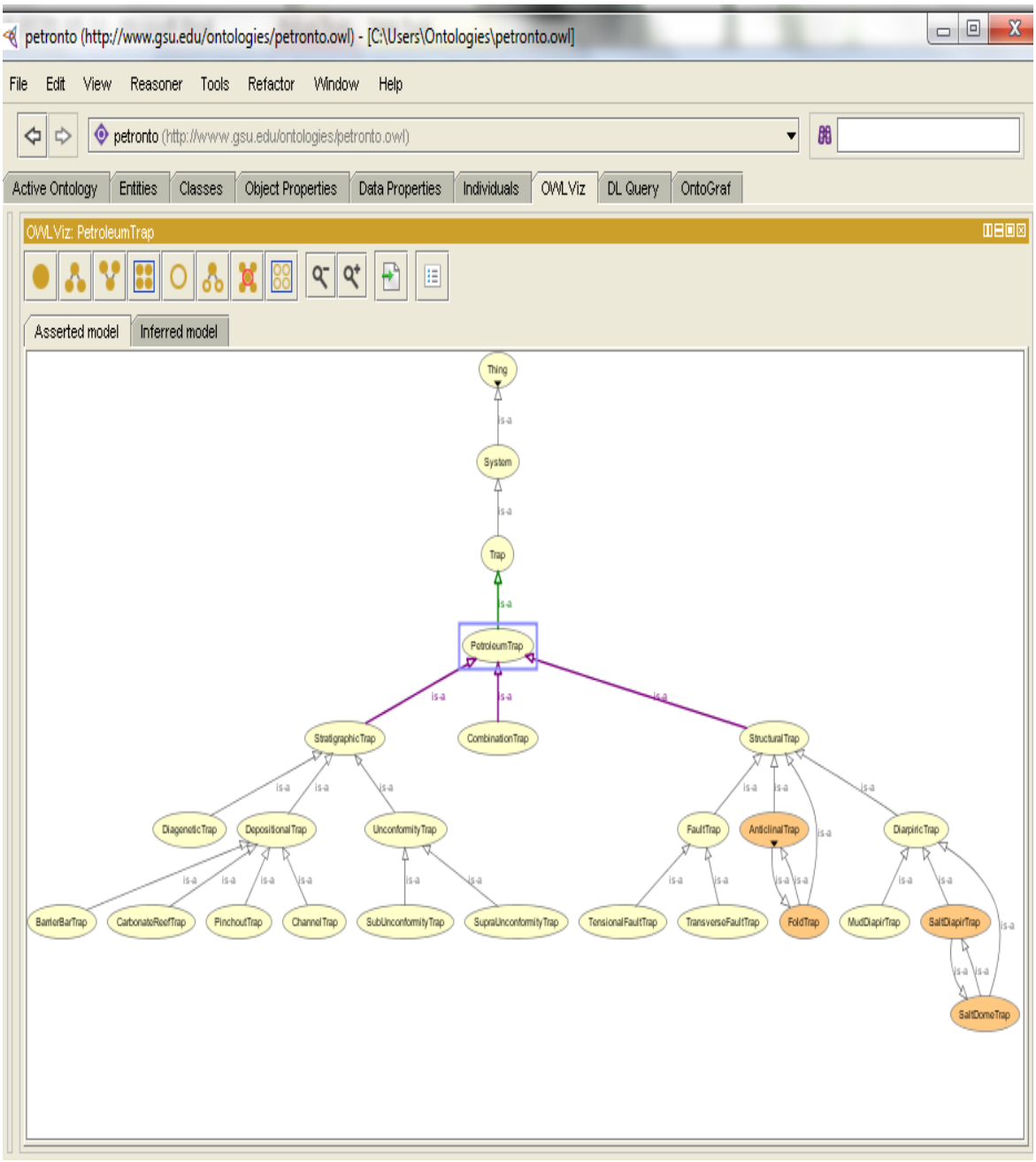

Figure 7.9. Screen shot showing the hierarchical relations among kinds of PetroleumTrap.

Relevant Codes:

List 7.15 Code showing the construction of the SourceRock, PetroleumReservoir and CapRock as disjoint classes.

```
 <rdf:Description>
       <rdf:type rdf:resource="&owl;AllDisjointClasses"/>
       <owl:members rdf:parseType="Collection">
          <rdf:Description rdf:about="&petronto;CapRock"/>
          <rdf:Description rdf:about="&petronto;PetroleumReservoir"/>
          <rdf:Description rdf:about="&petronto;SourceRock"/>
       </owl:members>
    </rdf:Description>
```

List 7.16 Code showing the construction of PetroleumTrap from the disjoint union of the SourceRock, PetroleumReservoir and CapRock classes.

```
<!-- http://www.gsu.edu/ontologies/petronto.owl/PetroleumTrap -->
    <owl:Class rdf:about="&petronto;PetroleumTrap">
       <rdfs:subClassOf rdf:resource="&petronto;Trap"/>
         <owl:disjointUnionOf rdf:parseType="Collection">
              <rdf:Description rdf:about="&petronto;CapRock"/>
              <rdf:Description rdf:about="&petronto;ReservoirRock"/>
              <rdf:Description rdf:about="&petronto;SourceRock"/>
         </owl:disjointUnionOf>
    </owl:Class>
```

List 7.17 Code showing the construction of CombinationTrap from the union of the StratigraphicTrap and StructuralTrap classes. Again the code shows that CombinationTrap is a subclass of PetroleumTrap.

```
<!-- http://www.gsu.edu/ontologies/petronto.owl/CombinationTrap -->


  <owl:Class rdf:about="&petronto;CombinationTrap">
    <owl:equivalentClass>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <rdf:Description rdf:about="&petronto;StratigraphicTrap"/>
          <rdf:Description rdf:about="&petronto;StructuralTrap"/>
        </owl:unionOf>
      </owl:Class>
    </owl:equivalentClass>
   <rdfs:subClassOf rdf:resource="&petronto;PetroleumTrap"/>
 </owl:Class>
```

7.5 **Extensions made to *humanCommerce.owl***

Concepts in the *humanCommerce.owl* file include Dig, Drill, Mining, Extraction, Exploration, ResourceExtraction, and Production.

7.5.1 A new class PetroleumExtraction is created as a subclass of Extraction

The *humanCommerce.owl* file has the class Production under which is placed the class EngineeringActivity as a subclass. EngineeringActivity further has a subclass Extraction, and Extraction has the subclass ResourceExtraction.

A new class PetroleumExtraction is created as a subclass of ResourceExtraction. PetroleumExtraction is also made equivalent to PetroleumProduction and specialized as PrimaryRecovery, SecondaryRecovery, and TertiaryRecovery. PrimaryRecovery is made equivalent to PrimaryProduction, and TertiaryRecovery is made equivalent to EnhancedOilRecovery.

SecondaryRecovery is specialized as GasFlooding and WaterFlooding. GasInjection is created as an equivalent class of GasFlooding, and WaterInjection is similarly created as an equivalent class of WaterFlooding.

New classes MiscibleGasInjection, Thermal EOR, and ChemicalFlooding are created as subclasses of EnhancedOilRecovery. SteamInjection and HotWaterInjection are made subclasses of Thermal EOR. Further, SteamInjection is made equivalent to SteamFlooding. ChemicalFlooding is made equivalent to ChemicalEOR. DetergentInjection and AlkalineFlooding are made subclasses of ChemicalEOR. DetergentInjection is made equivalent to Micellar-PolymerFlooding.

These recovery methods involve certain reservoir drive mechanisms, which are either natural drive mechanisms or artificial drive mechanisms. The recovery methods are correspondingly connected to the classes NaturalDriveMechanism and ArtificialDriveMechanism through object properties.

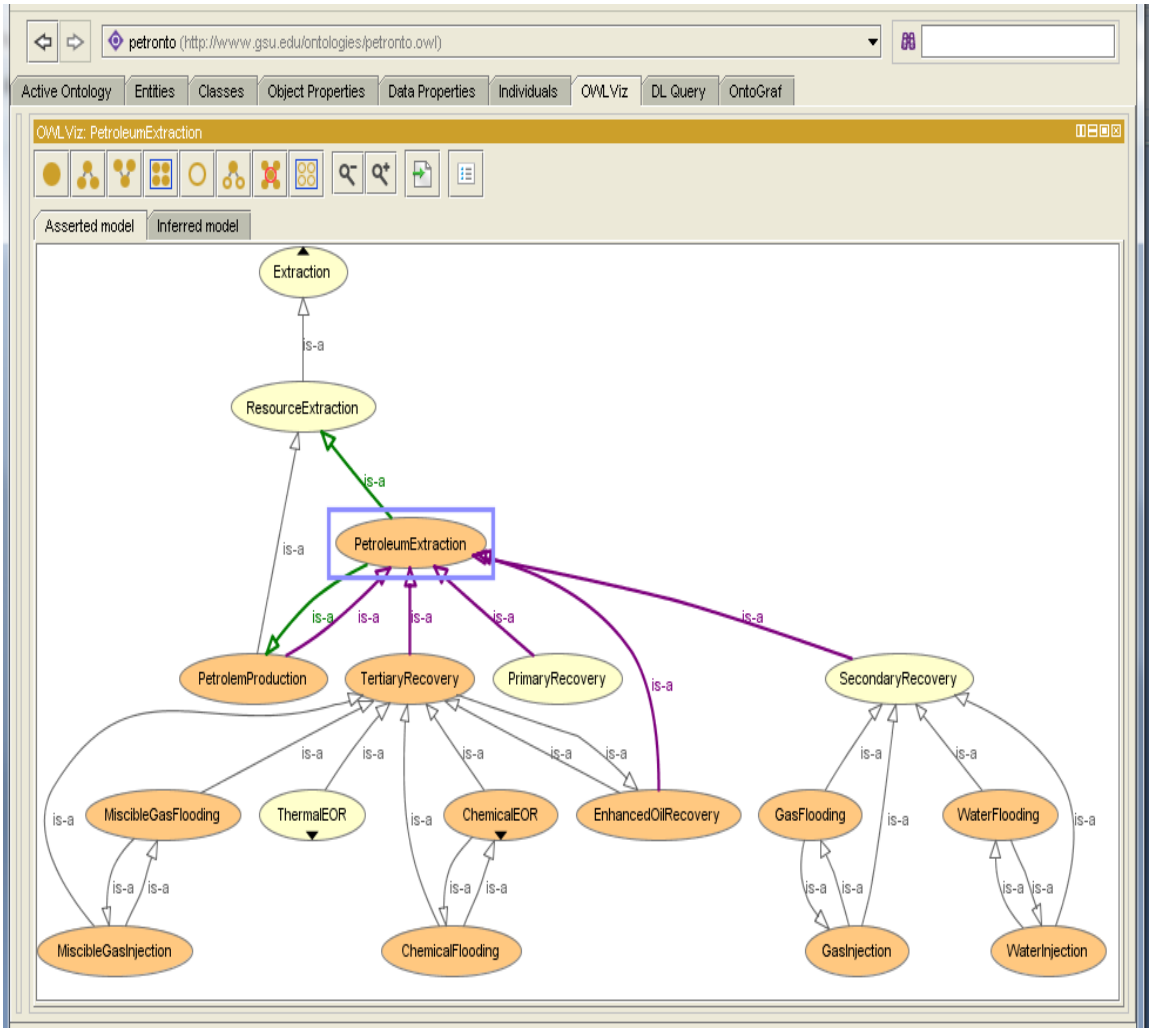

Figure 7.10. Screen shot showing superclasses and subclasses of PetroleumExtraction.

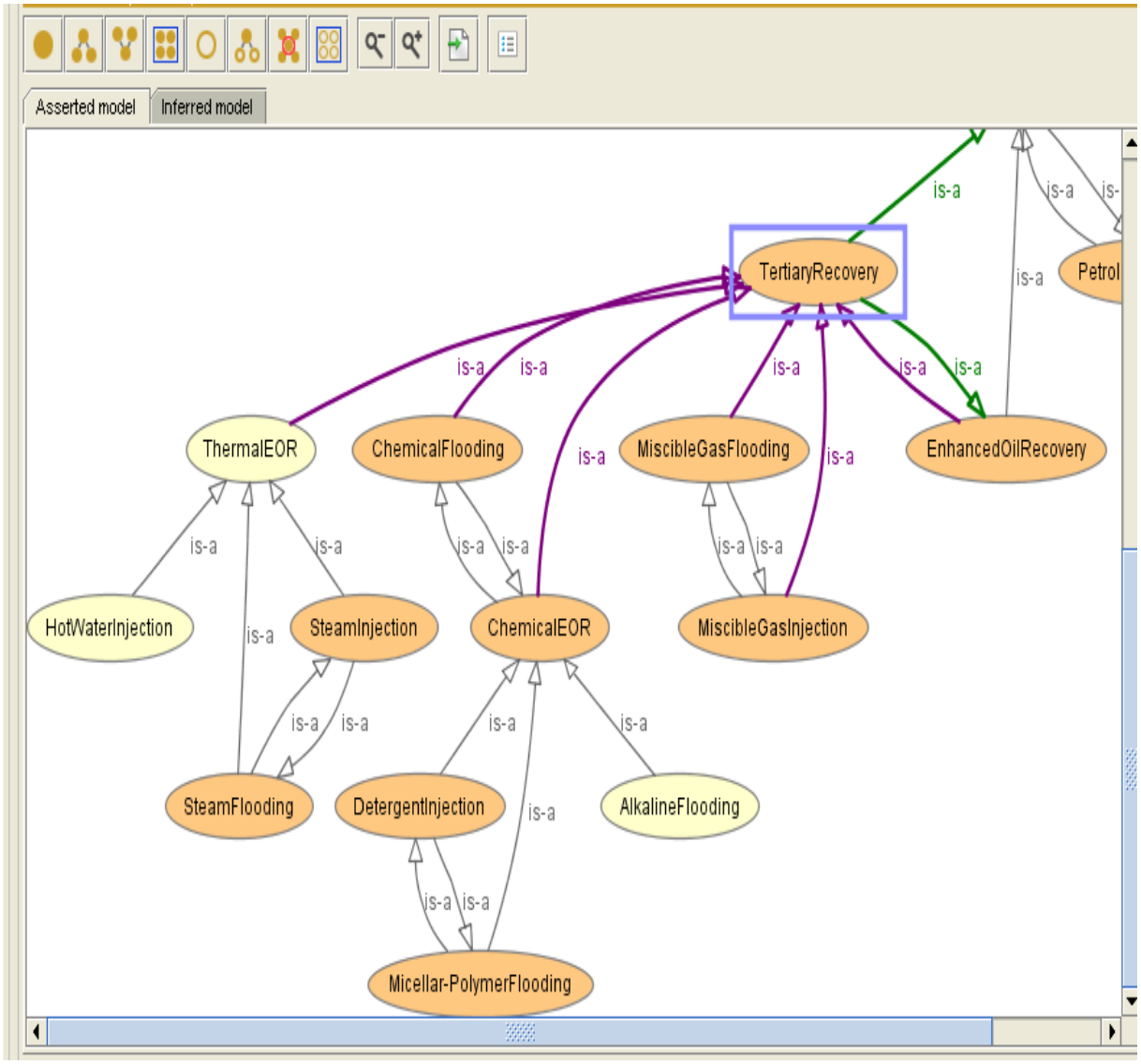

Figure 7.11. Screenshot showing subclasses of TertiaryRecovery

Relevant Codes

List 7.17 Code showing the description of PrimaryRecovery, SecondaryRecovery, and TertiaryRecovery as disjoint classes

```
<rdf:Description>
    <rdf:type rdf:resource="&owl;AllDisjointClasses"/>
      <owl:members rdf:parseType="Collection">
        <rdf:Description rdf:about="&petronto;PrimaryRecovery"/>
        <rdf:Description rdf:about="&petronto;SecondaryRecovery"/>
        <rdf:Description rdf:about="&petronto;TertiaryRecovery"/>
      </owl:members>
</rdf:Description>
```

List 7.18 Code showing the description of ChemicalEOR, MiscibleGasInjection, and ThermalEOR as disjoint classes

```
<rdf:Description>
    <rdf:type rdf:resource="&owl;AllDisjointClasses"/>
    <owl:members rdf:parseType="Collection">
        <rdf:Description rdf:about="&petronto;ChemicalEOR"/>
        <rdf:Description rdf:about="&petronto;MiscibleGasInjection"/>
        <rdf:Description rdf:about="&petronto;ThermalEOR"/>
    </owl:members>
</rdf:Description>
```

7.6 Extensions to *phenFluidDynamics.owl*

This file, *phenFluidDynamics.owl* contain classes and properties relating fluid dynamics. Concepts in this file include Eddy, CapillaryAction, Jet, and Flow.

7.6.1 Multiple new classes added to the class FluidPhenomena

The *phenFluidDynamics.owl* file contains the class FluidPhenomena which is a subclass of the Phenomena located in the *phen.owl* file.

A new class PetroleumDriveMechanism is created as a subclass of FluidPhenomena. PetroleumProductionMechanism is also created as an equivalent class of PetroleumDriveMechanism.

PetroleumDriveMechanism is specialized as NaturalDriveMechanism and ArtificialDriveMechanism. NaturalDriveMechanism is declared equivalent to PrimaryProductionDriveMechanism and PrimaryRecoveryDriveMechanism.

The following classes: WaterDrive, GasCapDrive, DissolvedGasDrive and GravityDrive are made subclasses of NaturalDriveMechanism.

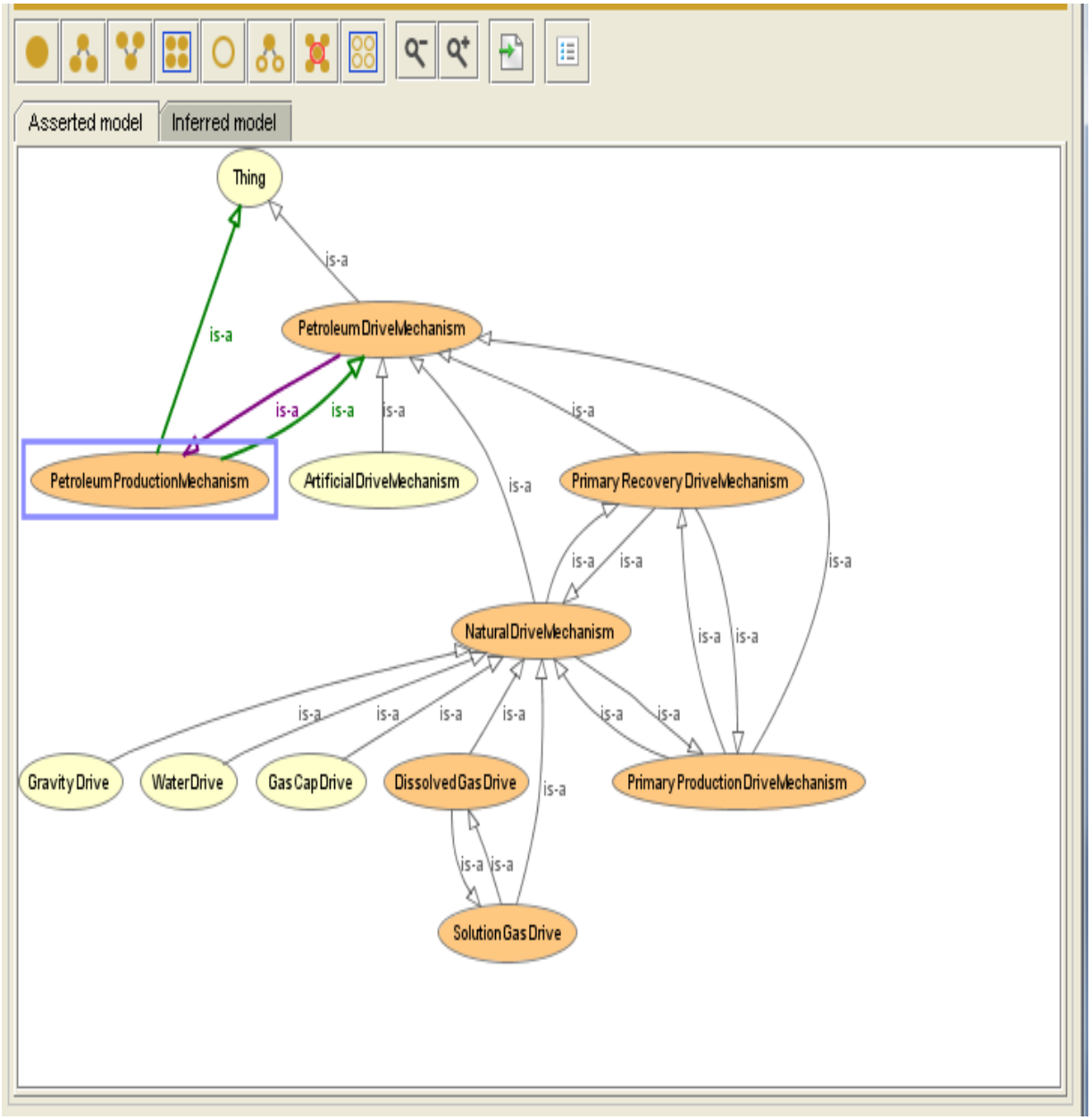

Figure 7.12. Screen shot showing the relationships in the PetroleumProductionMechanism class.

Relevant code

List 7.19 Code showing the description of DissolvedGasDrive, GasCapDrive, GravityDrive and WaterDrive as disjoint classes.

```
<rdf:Description>
    <rdf:type rdf:resource="&owl;AllDisjointClasses"/>
    <owl:members rdf:parseType="Collection">
        <rdf:Description rdf:about="&petronto;DissolvedGasDrive"/>
        <rdf:Description rdf:about="&petronto;GasCapDrive"/>
        <rdf:Description rdf:about="&petronto;GravityDrive"/>
        <rdf:Description rdf:about="&petronto;WaterDrive"/>
    </owl:members>
</rdf:Description>
```

CHAPTER 8

CONCLUSION

This Petroleum Geology ontology captures more concepts (terms) commonly used in the Petroleum Geology domain, than SWEET ontologies. These concepts modeled, especially pertains to types of petroleum, petroleum reservoirs, petroleum traps and the different drive mechanisms associated with petroleum recovery or production. These concepts have rich relationships (objects and datatype properties) with other relevant concepts modeled in this ontology. Concepts that are equivalent or disjont are clearly shown.

In the context of reuse, this ontology will serve as a valuable source of information for researchers in this area of Petroleum Geology as well as provide greater clarity of understanding for computer scientists in knowledge modeling involving this area of Geology. Experts in the Petroleum Geology domain can quickly refer to the concepts in this extended ontology and the relationships between these concepts to map between related data sets. This ontology may also be integrated with other existing ontologies on other aspects of the Petroleum Geology domain to form a larger ontology. The RDF graph of this ontology may be further extended to include additional or more specific concepts than is presently covered in this ontology.

As a scientific knowledge representation, the ontology captures a broad range of key concepts which will greatly reduce the burden of knowledge seeking and therefore enhance the process of knowledge discovery for users in this domain. The ontology may be mapped to a relational database schema in which the table in a database may be considered a class in the ontolology. The fields in the table will then equal the predicates in the ontology, and each cell in the table will equal an object in the ontology. Every statement in the RDF triple, subject-

predicate-object is similar to a value in a cell of the database table. Similarly, when a relational database table is converted to an RDF triple, each cell in the table converts to one RDF triple. In such cases the best practice is to design a URI for the table, with a prefix. Each row is then identified by concatenating the table name with the ID of each row. The fields may also be uniquely identified by concatenating the table name with the column name. A knowledge base may be built on this ontology by creating instances, that is, by providing values for the classes and properties. The knowledge base can be put on the web to receive data from petroleum geologists. This however requires building a web application which is beyond the scope of this thesis. The novelty of this ontology is indeed a helpful resource. Given the standard RDF data model used in making the ontology, the related knowledge base can be integrated with others, with the help of the Semantic Web browsers and search engines, to allow users to search and retrieve very refined information from several sources.

REFERENCES

Alexander, P. (2011)."Ontologies: Relational Vocabularies." In *The MMI Guides: Navigating the World of Marine Metadata*. http://marinemetadata.org/guides/vocabs/ont, (June, 2011).

Allemang, D. and Hendler, J. (2008). Semantic Web for the Working Ontologist. Effective Modelling in RDFS and OWL. United States, Morgan Kaufmann Publishers, 330 p.

Baader, F., Calvanese, D., McGuineness, D., Nardi, D., Patel-Schneider, P. (2003). Editors. *The Description Logic Handbook*. Cambridge University Press.

Babaie, H. (2011). Modeling geodynamic processes with ontologies. Book Chapter in the '*Geoinformatics: Cyberinfrastructure for the Solid Earth Sciences*'' Randy Keller and Chaitan Baru (eds), Cambridge Press, 166-189.

Babaie, H. (2011). "Ontological relations and spatial reasoning in Earth science ontologies". For: *Societal Challenges and Geoinformatics*. Krishna Sinha, David Arctur, Ian Jackson, and Linda Gundersen (eds), Geological Society of America (GSA) Special Paper 482, 13-27.

Babaie, H. A. Oldow, J. S. Babaei, A. Ave Lallement, H. G. and Watkinson, A. J. (2006). Designing a modular architecture for the structural geology ontology. In *Geoinformatics: Data to Knowledge*, ed. A. K. Sinha. *Geological Society of America Special Paper* **397**, 269-282.

Berners-Lee, T., Hendler, J., Lassila, O. (2001). The Semantic Web. *Scientific American*, 284(5):34 – 43.

Berners-Lee, T. and Kagal L. (2008). The Fractal Nature of the Semantic Web. http://dig.csail.mit.edu/2007/Papers/AIMagazine/fractal-paper.pdf, (July, 2011).

Boggs, S. Jr. (2006) Principles of Sedimentology and Stratigraphy, 4th ed.: Pearson Prentice Hall, New Jersey, 662p.

Chikofsky, E.J., Cross J.H. (January 1990). "*Reverse Engineering and Design Recovery: A Taxonomy in IEEE Software*". *IEEE Computer Society*: pp. 13–17.

England, W.A. (1994). Secondary migration and accumulation of hydrocarbons. AAPG Mem. 60, 211-217.

Ellson, J. and North, S. (2005). Graphviz – Graph Visualization Software http://www.graphviz.org, (July, 2011).

Gennari, J., Musen, M., Fergerson, R., Grosso, W., Crub´ezy, M., Eriksson, H., Noy,N., and Tu, S. (2003). The evolution of Prot´eg´e-2000: An environment for knowledge-based systems development. International Journal of Human-Computer Studies, 58(1):89–123.

Ghazvinian, A., Noy, N.F., Musen M. A. (2010). How Orthogonal are the OBO Foundry Ontologies?

Gomez-Perez, A., Rojas-Amaya, D. (1999). Ontological Reengineering for Reuse. EKAW'99, LNAI 1621, 139-156.

Graybeal, J., Alexander, P. (2011). "What is an Ontology?." In *The MMI Guides: Navigating the World of Marine Metadata*. http://marinemetadata.org/guides/vocabs/ont/definition, (June, 2011).

Gruber, T. (1993). A Translation Approach to Portable Ontology Specification, Knowledge Acquisition 5(2), 199-220.

Gruniger, M. and Fox, M.S. (1995). Methodology for the Design and Evaluation of ntologies. In: *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing*, IJCAI-95, Montreal.

Hebeler, J., Fisher, M., Blace, R., Perez-Lopez, A. (2009). Semantic Web Programming. Wiley Publishing, Inc., Indianapolis, 616p.

Heflin J. (2009). "OWL Web Ontology Use Cases and Requirements" http://www.w3.org/TR/webont-req, (August, 2011).

Hyne, N.J. (2001). Nontechnical Guide to Petroleum Geology, Exploration, Drilling, and Production. 2nd Edition: PennWell Corporation, Tulsa Oklahoma,563p.

Horridge, M. and Brandt, S., (2011). A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools. Edition1.3, The University of Manchester.

Horrocks, I., Patel-Schneider, P. F., and F. van Harmelen (2003). From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*,1(1).

Hunt, J.M. (1996). Petroleum Geochemistry and Geology. 2nd Ed.: W.H. Freeman, New York, 743p.

Knublauch, H., Musen M. A., Rector A.L. (2004). Editing Description Logic Ontologies with the Protégé OWL Plugin International Workshop on Description Logics, Whistler, BC Canada.

Loh, S., Litchtnow, D., Borges, T., Piltcher G. (2008). Evaluating the Construction of Domain Ontologies for Recommender Systems Based on Texts.

McGuiness, D.L. and Harmelen, F.V. (2004). OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features, (August, 2011).

Murck B.W., Skinner, B.J., Porter, S.C. (1996). Environmental Geology. John Wiley and Sons, Inc. New York, 535p.

Neiswender, C. (2009). "What is a Controlled Vocabulary?." In The *MMI Guides: Navigating the World of Marine Metadata.* http://marinemetadata.org/guides/vocabs/vocdef, (June, 2011).

Neiswender, C., Miller, S.P., Bermudez, L., Montgomery, E., Isenor, A. (2011). "Vocabularies: Dictionaries, Ontologies, and More." In The *MMI Guides: Navigating the World of Marine Metadata.* http://marinemetadata.org/guides/vocabs, (June, 2011).

Neiswender, C., Montgomery, E., (2009). "Metadata Interopearability-What Is It, and Why Is It Important. In The *MMI Guides: Navigating the World of Marine Metadata.* http://marinemetadata.org/guides/mdatainteroperability, (July, 2011).

Noy, F.N., McGuinness, D.L. (2003). *Ontology development 101: A guide to create your first ontology*. http://ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinnes.doc, (March, 2011).

Raskin, R. (2005a). Semantic Web for Earth and Environmental Terminologies (SWEET) Ontologies. http://sweet.jpl.nasa.gov/ontology, (April, 2011).

Raskin, R. (2005b). Guide to SWEET Ontologies. http://sweet.jpl.nasa.gov/guide.doc (April, 2011)

Raskin, R. (2006). *"Development of Ontologies for Earth System Science.* Geological Society Of America Special Paper 397.

Raskin, R. and Pan, M.J. (2005). *"Knowledge representation in the Semantic Web for Earth and Environmental Terminology (SWEET)*." Computers & Geosciences 31, 1119-1125.

Selley, R.C. (1998). Elements of Petroleum Geology. 2nd Edition: Academic Press, California, 470p.

Showalter, T.T. (1979). Mechanics of secondary hydrocarbon migration and entrapment. AAPG Bull. 63, 723-760.

Tissot, B.P. (1977). The application of the results of organic chemical studies in oil and gas exploration. In “Developments in Petroleum Geology” (G.D. Hobson, ed.), Vol. 1, pp. 53-82. Applied Science Publishers, London.

Uschold, M., Gruninger, M. (1996). Ontologies: "Principles, methods and applications. *Knowledge Engineering Review, 11(2).*

Warden, R. (1992). *Software Reuse and Reverse Engineering in Practice*. London, England: Chapman & Hall. pp. 283–305.

Yu, L. (2011). A Developers Guide to the Semantic Web. Springer Heidelberg Dordredcht, NewYork, 608p.