ONTOLOGY MODELING
the software engineering way

Siva Kumar Inguva

Advisor: Dr. Lillian Cassel

Completed: Spring ’2011

This thesis final report is submitted in partial fulfillment of the requirements for the degree of

M.S. in Computer Science
And
M.S. in Software Engineering

Department of Computing Sciences
Villanova University
Villanova, Pennsylvania
U.S.A.

ACKNOWLEDGEMENTS

First and foremost, I would like to offer my sincerest gratitude to my supervisor Dr. Lillian (Boots) Cassel for giving me this opportunity to study and explore Ontologies. This certainly has given me a scope to push my boundaries to enter into new realms of Computing Science. I am grateful for all the support and guidance given to me without which the project would never be in this shape.

I am grateful to Dr. Jan Buzydlowski, Dr. Bob Siegfried and other Villanova research group members for their valuable inputs, encouragement and constant support.

Last but not least, I would like to thank the Villanova University and in particular the Department of Computing Sciences for giving me this wonderful opportunity to explore and present my views on Ontology.

Siva Kumar Inguva
(April '20, 2011)

Contents

Abstract

In recent years, data or information has increased exponentially. This data or information is readily available online for consumers to read, understand and reproduce according to their needs. But the question(s) is/are can technologies sustain this information overload? How much time does it take to process a search query? How accurate are the results? How complex is the query? Can system understand what it is looking for? and so on.

The solution to address this complexity is to make data smart enough for system to understand process and get accurate results. Semantic Web is about adding semantics to the data so that the information online is machine readable. One way of realizing Semantic web is to connect data with proper relations or properties forming an *ontology*.

Ontologies are the back bone of semantic web. This paper gives a brief introduction to semantic web, techniques and technology involved in it and finally focuses on ontology and ways to model ontologies. We propose a new way of ontology modeling called Ontology Development Life Cycle (ODLC) which inherits its principles from Software Development Life Cycle (SDLC). The paper also explains *Computing Ontology* that is created using ODLC. Computing Ontology is an ontology that describes various disciplines, topics, and subtopics that belong to the domain of Computing Sciences. It also describes the ways in which these topics and subtopics are related to each other and to various sub disciplines. Creating and maintaining a computing ontology can come in handy for various real time applications such as Curriculum Developments, Indexing of Digital Libraries, and supporting searching.

APPINDEX I – List of Examples

APPINDEX II - List of Figures

APPINDEX III – List of Tables

1. Introduction

The world of information or information space is never the same. It changes every minute, every second and even in every n^th of a second if we could measure it, and major percentage of these changes occur on the 'World Wide Web'. Going back to the past when there was no Internet, no emails, and no online transfer of data, information remained and confined to one computer and the only means of data transfer is through a disk, it is the necessity for data transfer made Sir Tim Berners-Lee to propose "World Wide Web" or in short WWW[1][2]. The current web is majorly inferred by people or specific software written to work with that data. But the technological advancements demand for computers to understand and infer the data on web just like the way humans do, that is to obtain, read and understand data. This gave heads up to **Semantic Web** [3]**.** The semantic web is built upon **Ontologies**: *the backbone of the semantic web*.

Ontology is a description of concepts and relationships that can exist for an object or a community of objects. They are designed to enable sharing and reuse of information and knowledge. Ontology modeling could be extremely complicated; it needs for ontology engineers who are knowledge experts, domain experts, software engineers. For decades, Software Development Life Cycle or popularly known as SDLC is treated as best way of developing software, the question is can ontology be developed as a software using software engineering design processes/techniques or SDLC's? To bridge this gap a few proposals were made in the past, to use Unified Modeling Language (UML) as a building block to design ontologies [4] but the noticeable difference between UML design with respect to ontology modeling is UML is compact and is used by human experts while ontologies being data centric could vary from small to extremely large in both size and components and has be machine readable that is read by an intelligent system.

Ontologies being an increasingly important topic for research and development can also be developed using raw (non SDLC) methods like - building from the scratch, re-engineering of available ontology, fusion or merging of two or more ontologies etc.
The purpose of this paper is to study existing methodologies (both SE and non-SE) of modeling ontologies. The study is organized in to 6 sections. Section 2 gives an introduction to Semantic Web and focuses on how and why semantic web is a solution for issues on information space. Semantic web languages like RDF, OWL, DAML+ OIL helps in realizing the very concept are explained in sub-sections. Section 3 gives a brief idea on ontologies and its purpose to realize structured information space. Section 4 provides a comparative evaluation on general ontology development methodologies. A comparison and contrast on SDLC with Ontology development life cycle and features of ODLC are explained in its sub-sections. Section 4 also focuses on ontology building tools like Protégé, and visualization techniques, and finally full scale ontology is explained adopting Computing Ontology [4] as an example in section 5.

1.1 Related Work

This section gives a brief description on scholarly articles related to this paper. These wonderful works have always motivated us to take a step further than what we originally planned for.

1. Software Engineering Ontology for Software Engineering Knowledge Management in Multi-site Software Development Environment.

- The above paper proposes Software Engineering Ontology that represents software engineering knowledge in ontology and whose instantiations are representing evolution of that project or simply the project data.
- The advantage of such Software Engineering Ontology is, easy to recall all the necessary details and relations among the project data just by parsing the ontology.
- This paper triggered us to look at the other way round, that is how to develop an ontology using Software Engineering techniques
- More details about the article can be found in reference section - [5]

2. Software Engineering Ontology - A Development Methodology

- This paper talks about SWEBOK - Software Engineering Body of Knowledge which is proposed with 5 objectives one of them being - To characterize the contents of Software Engineering discipline.
- This paper provides a literature review of ontology development methodologies which helped in learning about the already existing ontology modeling techniques.
- More details about the article can be found in reference section - [6]

3. Ontology as Software Artifact

- This paper describes ontologies as software artifacts which could be
  - Ontologies as Software Artifacts at Development Time
  - Ontologies as Software Artifact at Run Time
- Gives a high level idea on which level of software engineering detail (phase) could use ontology for software development.
- More details about the article can be found in reference section - [7]

4. Benchmarking Criteria to evaluate Ontology building methodologies

- This paper describes ontology as a part of software product and hence proposes that SDLC can be used to create these parts.
- Proposes various methodologies to build ontologies *viz* Software life cycle model process, Project management process, Software development oriented process, Integral process.

- This paper helped in understanding the skeleton view of ontologies - outer requirements Vs. Data centric structures.
- More details about the article can be found in reference section - [8]

5. A Software Engineering approach to ontology building
- This paper proposes a software engineering process to build ontologies - UPON.
- UPON is a Unified Process for Ontology building. This is very closely related to our study.
- This paper helps in understanding on how to map SD processes with OWL components.
- More details about the article can be found in reference section - [9]

6. Applications of ontologies in Software Engineering
- This paper describes on how ontologies can be applied for software development.
- Every stage of a software development life cycle is analyzed from an ontological perspective. That is SDLC with respect to ontolo gy which is kind of the other way round to our study (Ontology with respect to SDLC).
- This paper helped in understanding SDLC's from an ontology stand point.
- More details about the article can be found in reference section - [10]

7. Semantic Web Languages – Strengths and Weakness
- This paper gives information on languages which help in realizing Semantic Webs.
- This paper helped in understanding the settle differences between the languages from a comparison perspective.
- More details about the article can be found in the reference section - [11]

1.2 From web 2.0 to Web 3.0:

Before looking at what semantic web is about all let us start with the classifications of WEB. Nova Spivak has proposed a chart of web evolution which essentially describes the growth of World Wide Web and its technologies over a period of time. Web 1.0 is linear and primitive where a web expert controls the webpage's which are created by other experts and organizations. Web 1.0 is a read-only form of web targeted for users and customers to browse and learn information and products. The current web is Web 2.0, which is mainly considered as user centric web and deals with the social networks, web applications, and technologies for data transfer and information management, it focuses on social and business usage of web and its technologies. Web 3.0 is data centric, where the focus is to make data smart by adding semantics to the data. This era of web contains knowledge bases with well structured information and Artificial Intelligent agents capable of interpretation this information. There is no bright line to differentiate between Web 1.0, Web 2.0 and Web 3.0 and no one will be able to make this decision. The reason for versions is merely to understand the related technological advancements.

The semantic web is treated as Web 3.0 wherein the technologies are data centric and automation of information representation, querying and sharing using technologies like SPARQL, RDF/OWL, AJAX, Cloud Computing and Ontologies.

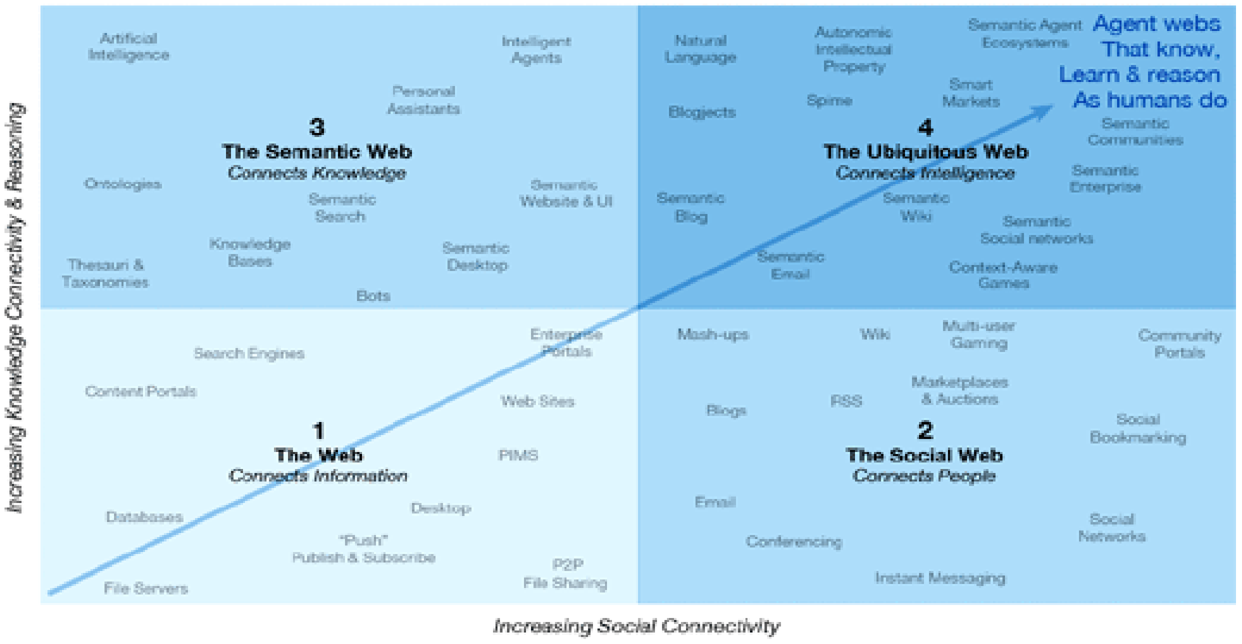


Fig-1.1 Evolution of Internet.[12]

2. Semantic Web

Semantic Web: "the semantic web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation." [2]
Semantic web is like a mesh of all information related to each other through relationships or otherwise called properties. The purpose of relating information is to keep data structured and allow querying. Consider a situation like 'you want to find a pizza place that serves a particular variety of pizza with a particular kind of sauce and a special kind of topping' which needs multiple information dots to be connected which is expensive on current web infrastructure with

heavy overloading of information, while semantic web already has information represented in this form, thus making a user's life easier.

2.1 Information Representation

Information Representation is a complex process. Expressiveness of information varies on type of the user or consumer of that information and different consumers demand information for different purposes, also different people describe same data in different ways which increases the complexity of information 10 fold.

Therefore, information producers (authors, developers) must produce information that could minimize consumer's efforts while avoiding redundancy of information, strictly No Duplications. The question still remains, how can information representation be done? Looking this in the line of an IT development system the most important phase of any software development is requirement analysis so the most required items for information representation is making the information available which is abundant on current web at various levels of expertise. The next challenge is to harvest and encode this information dynamically to leverage the semantics and create connection amongst this information. The information represented is well defined and linked for machine to navigate through its parts hence machine readable.

2.2 Connecting the information space

Semantic web is about leveraging current web infrastructure with more connected, structured and widely distributed body of knowledge with the targeted advantage of minimizing human efforts. This connected or structured information is very specific to a domain, scope, or the type or kind of relationship the information is connected with. These specifications are called semantics and are defined though a vocabulary of semantic languages like

- Universal Resource Identifiers (URI)
- Extensible Markup Language (XML),
- Resource Description Framework (RDF),
- Resource Description Framework Schema (RDFS),
- DAPRA Agent Markup Language + Ontology Interface Layer (DAML + OIL)
- Web Ontology Language (OWL). [13]

Tim Berners Lee proposed the way to add semantics to the data is through a language tower or layered architecture [14] described in figure Fig-2.2.1.

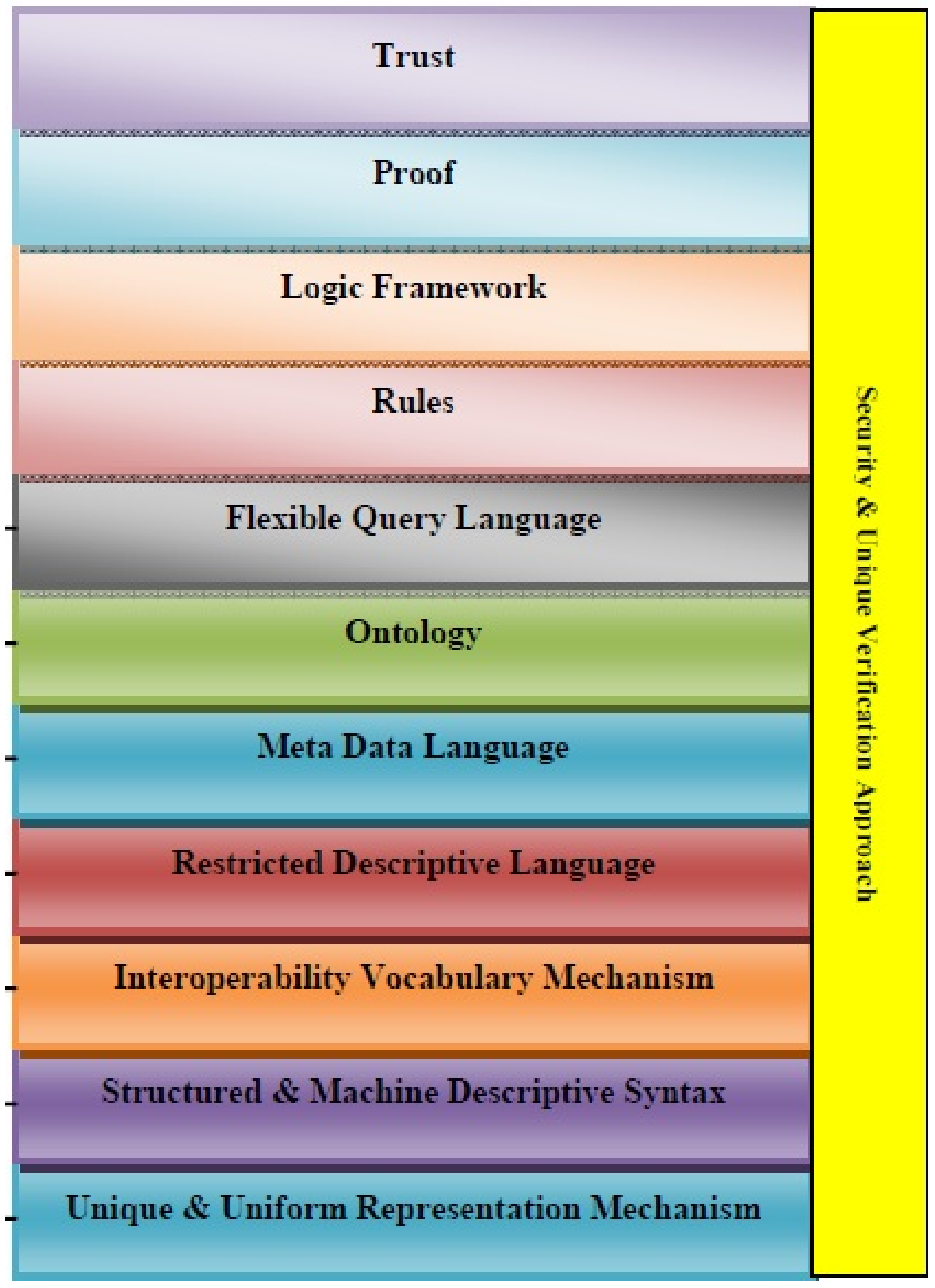

Trust
Proof
Logic Framework
Rules
Flexible Query Language
Ontology
Meta Data Language
Restricted Descriptive Language
Interoperability Vocabulary Mechanism
Structured & Machine Descriptive Syntax
Unique & Uniform Representation Mechanism
Security & Unique Verification Approach

Fig-2.2.1 Tower or Layered Architecture *(reproduced form [14])*

In general the architectural layered models are defined with the idea of inter operable components layered in the order of their dependencies. Architectures like Network architecture layers or peer to peer architectural layers are defined where the functionality of components of one layer is dependent on the functionality of other layer below it, but it is not the same in case of semantic web architecture. The technologies defined can exist independently. The significance of adapting this structure is to show the machine processing ability of semantic web languages.

2.3 Semantic Web Languages

The semantic web languages are just not any languages, they are not aimed to pump the data to a storage data house and retrieve back when asked for, they are designed to inference data associated with them and create a semantic markup on the information they deal with.
The most desired feature of a semantic language is that it should be capable to design intuition of human users without losing the adequate expressive power. [15] It should have capabilities to define specific syntax and formal semantics while being compatible with current web standards.

2.3.1 URI – Universal Resource Identifier

URI is not a language but it is the most important aspect of any semantic structure. URI, the Universal Resource Identifier specifies the location of the resource. The resources to semantics are needed to be added, the very same resource which is to be connected with some other resource which essentially is represented by a different URI. Therefore, URIs could be referred as "information representation constructs" [16]. URIs has ability to represent any kind of information either physical or abstract, it acts as an identity. URIs can even represent classes, objects, properties depending on the type of resource.

URIs is of two types URL (Uniform Resource Locator) and URN (Uniform Resource Name) are well known in the information spatial world.

- URLs: the Uniform Resource Locator's are network resolvable, meaning they demand for a network location to identify the resources. In other words they must have a network to retrieve resource.
- URNs: the Uniform Resource Names are another kind of URIs which provides global unique persistent identifiers.

All these URIs which represent a resource are constant, but resources change. New information may be added or old resources could be replaced in which the URI will have to point to the latest resource. This is where URI references are used. A URI ref is an extended URI that reference more specific resource. The idea is to keep the URI constant while providing a relative reference to the resources. This reference is separated by the character # called as fragment identifier.

URI identifies resource and the characters # identifies a portion of resource.

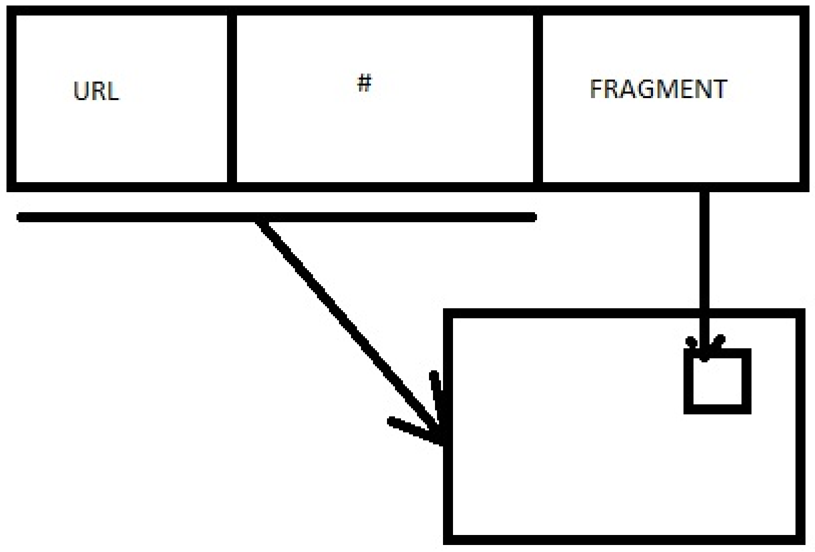


Fig-2.3.1.1 Structure of an URI

2.3.2 XML – Extensible Markup Language

Extensible Markup Language (XML) is considered to be a revolutionary language in the history of information technology. The purpose to create XML was to translate information into a form that can be interchanged and efficiently/economically transferred. “XML could be considered as a transport/syntax layer of the semantic architecture”. [17] XML is a text based language that could define other language, that is essentially XML is a meta language that allows users to create/define their own tags. It is adapted to define a structure of information to allow automated process of web content.

All the semantic languages follow XML like syntax. One of the outstanding features of XML is its ability to create well defined nested structure of information closely resembling to a hierarchical tree structure. Information is defined with a set of nodes, elements, attributes, properties and description tags. Parsers are available to process XML. A parser is software that navigates through the structure in xml and queries the information depending on the tags.
The following Example-2.3.2.1 describes a message sent to Tove form Jani.

```
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

Example-2.3.2.1- Message from Jani to Tove

There are various syntactic methods of defining an XML viz. a element centric approach and a attribute centric approach which are shown in Example-2.3.2.2 and Example-2.3.2.3

Element centric:

```
<myride>
     <rideName> car </rideName>
     <price>25000</price>
</myride>
```

Example-2.3.2.2: Showing Element centric structure

Attribute centric:

```
<myride rideNmae="car" price=25000 />
```

Example-2.3.2.3: Showing Attribute Centric Structure

The advantage of an attribute centric document is its compactness while the element centric is easily parsable.

Even with all these abilities, XML is not suitable for semantic web description. “The reason is XML defines syntax but not semantics” [18] and also the tags defined are in natural language for human understanding which are semantically ambiguous for a machine.

2.3.3 RDF – Resource Description Framework

As seen before, XML successfully creates a machine readable structure of information but fails to define semantics in a machine understandable way. In Example-2.3.2.1, machine does not understand who Tove or Jani are and what the massage is, for a machine it is just a set of binary information.

Resource Description Framework (RDF) fills the gap by adding meta-information. As the acronym says RDF is a resource descriptor. RDF provides a standard way to mark statement and resources with appropriate properties. These statements are powerful way of describing information.

Being a XML based language, RDF also captures the metadata of the documents, like the author, the creation date and treats them also as the resources as an assembly to the URL of the parent document. RDF can be treated as a semantic network which is a directed graph consisting of nodes and set of unidirectional edges. The nodes refer the concepts and the edges represent the properties of concepts. It consists of

- **Resources:** Resources are the basic building blocks of RDF which are associated with URIref. This URIref could be referring to a web-page, a picture or a collection of resources.

- **Properties:** Properties are termed as predicated which define the attribute or relations used to describe a resource.

- **Statements:** A specific resource with a named property and value of that property together is called a statement.

These 3 parts put together are called triples (which can also be called as the subject, the predicate and the object respectively) which can be recursive meaning a statement can act as resource for another set of triples. The arrangement is shown in Fig-2.3.3.1. These Statements provided in RDF are not sufficient to define relationships between property and resource, this limitation is overcome by combining RDF with XML.

RDF/XML is a way of representing an RDF in an XML format using XML’s ability of tagging and creating name-spaces. RDF can be modeled to exchange the statements.

`<rdf:RDF>` is the RDF header that describes that the concept of the document should be interpreted as an RDF. The representation is XML though; Therefore, the beginning statement will look similar to

```
<rdf: RDF xmlns: rdf=" http:// www.w3.org/1999/02/22 rdf-syntax-ns#">
```

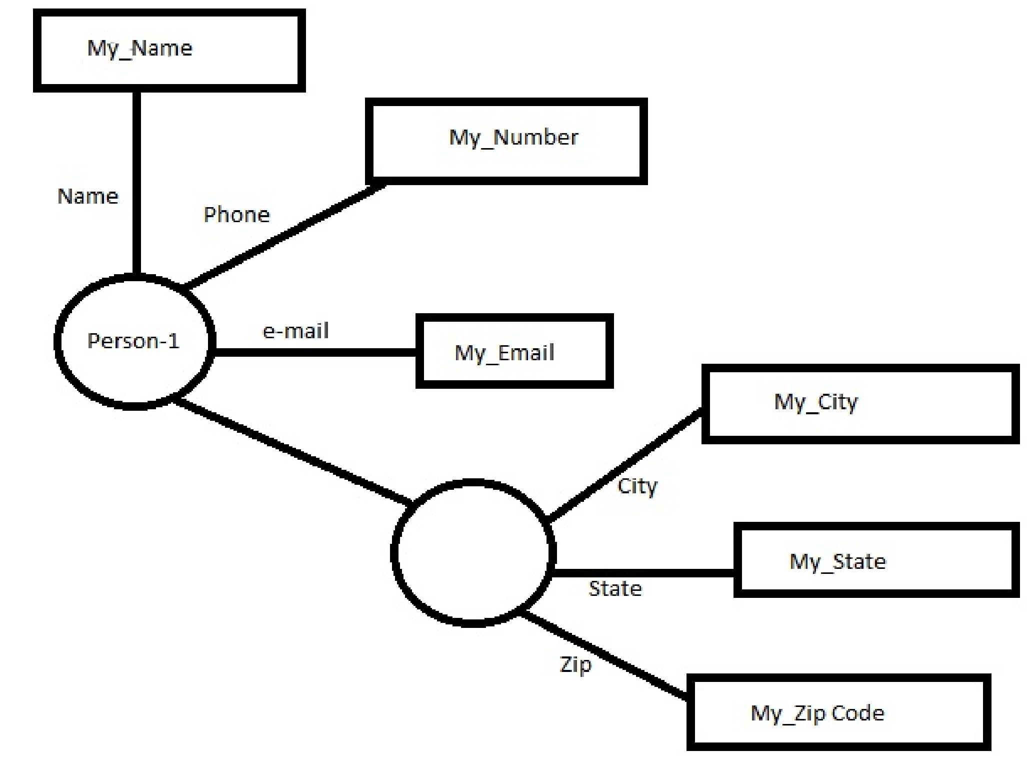


Fig-2.3.3.1 Subject-Predicate-Object structure of RDF.

The resources in an RDF/XML are generally named with convention similar to camel case like myPrice, menuDish and so on.

Statements in RDF/XML have the description tags as `<rdf: Description>` which provides the machine processable ability to the document.

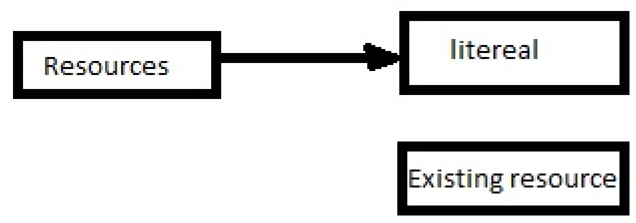


Fig-2.3.3.2 RDF Structure.

The subject can be an existing resource or a new resource.
RDF/XML allows representing a new resource using a tag called `<rdf: ID>`. It is an attribute whose value is URIref referencing from a location to another location.

```
<ref: Description ref: ID="resource URIref" />
```

An existing resource can be described using `<rdf: about>`

```
<rdf: Description rdf: about =" ID">
     <Property> property value </property>
</rdf: Description>
```

Examples:

New resource

```
<rdf: Description ref : ID ="my_interests" />
```

Existing resource

```
<rdf: Description rdf: about ="my_likes" >
  <fruit:favouritefruit>Apple</fruit: favouritefruit>
</rdf: Description>
```

RDF/XML also provides other descriptors like `<rdf: type>`, `<rdf: datatype>` to describe resource types.
Even with all these features RDF/XML still miss the cream of the semantics. Complex relations like Classes, Objects are needed to represent hierarchical structures.

2.3.4 OWL – Web Ontology Language

OWL the web ontology language is the best dish available for the party today. It is W3C recommendation. The concepts are defined as axioms and facts. It has ability to refer other ontologies as resources there by created a map of all available resources in well defined and well connected way.

Ontologies are defined as a formal specification of a shared conceptualization.

OWL has 3 layers

1. OWL-Lite: Contains a minimum number of concept-relation structures, less expressive and easy processing. It gathers its features from RDF(S). Owl-Lite is intended to create taxonomies.

```
<owl:Class rdfID="Fruit">
   <owl:interscetionOf rdf:parseType="Collection">
</owl:Class>
```

2. OWL-DL: A medium size of resource's related with other URIref's that fits in the Description Logic. All the primitives that are available in owl lite can be used in OWL-DL.

```
<owl:Class rdf:ID ="#Fruits">
     <owl:Restriction owl:cardibality="1">
          <owl:onProperty rdf:resource="#fruitName"/>
     <owl:Restriction/>
</owl:Class>
```

3. OWL-FULL: A complete vocabulary interpreted under simple constraints. Very expressive and perfect for machine processing. I

```
<owl:Class rdfID="Fruit">
  <owl:interscetionOf rdf:parseType="Collection">
    <owl:Class rdf:ID ="#Fruits">
       <owl:Restriction owl:cardibality="1">
          <owl:onProperty rdf:resource="#fruitName"/>
          <owl:allValuesFrom rdf:resource="some resource"/>
       </owl:Restriction>
       <owl:Restriction>
          <owl:hasValue rdf:datatype="some datatype:String">
          </owl:hasValue>
       </owl:Restriction>
     <owl:interscetionOf>
</owl:Class>
```

More about ontology and owl is described on Section-3 and 4 respectively.

2.4 Summary

Semantic web show a promising application for future.
1. One of the best customer served application involve searching, querying, finding and inferring information.
2. The original power of software lies when software harvests the content from diverse source and processes the information and structures the data.

3. Ontologies - What and Why

Ontology comes from the field of Philosophy that is concerned about study of existence. This concept is adapted by AI experts to represent existing knowledge in a declarative form.

In the field of Computing Science ontologies are defined as:

3.1 Definitions of Ontology

"An ontology defines the basic terms and relations comparing the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary"[19]

"An ontology is an explicit specification of a conceptualization"[19]

"Ontologies are defined as a formal specification of a shared conceptualization" [19]

"An ontology is a formal explicit specification of a shared conceptualization. Conceptualization refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon. Explicit means that the type concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that ontology should be machine readable. Shared reflects the notion that of an ontology captures consensual knowledge that is, it is not private of some individual but accepted by a group" [19]

"A logical theory which gives an explicit, partial account of a conceptualization" [19]

"A set of logical axioms designed to account for the intended meaning of a vocabulary" [19]

"An ontology provides the means for describing explicitly that conceptualization being the knowledge represented in the knowledge base" [19]

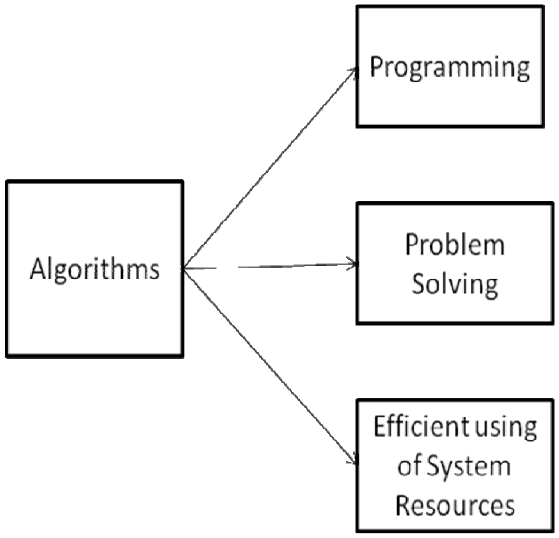
"An ontology is a hierarchically structured set of terms for describing a domain that can be used as a skeletal foundation for a knowledge base." [19]

All the above definitions look different in some or the other concept or perspective of defining the purpose of ontologies. But all these definitions commonly speak of "conceptualization" and "explicit". This section of the document describes the purpose and other features of ontologies.

3.2 Purpose of creating an Ontology

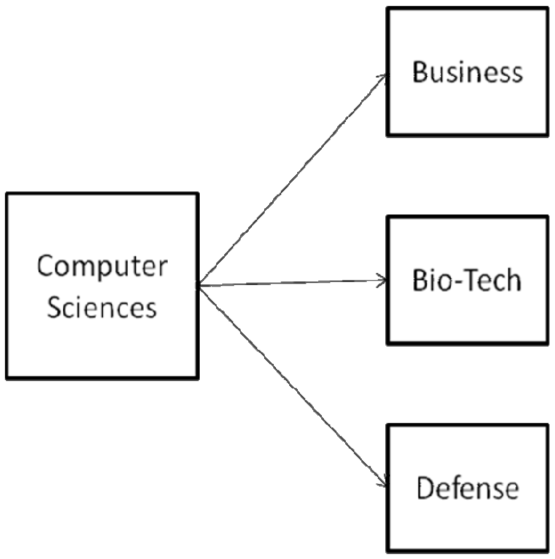
The purpose of creating an ontology is to [20]

- *To share common understanding of structures of information among people and software agents:* Information is repetitive same information could be available at many places and this duplication could be misleading. The role of ontologies is if these websites share a common ontology the agents/software agents/systems can extract and aggregate the information there by making the search efficient and also economic with respect to resources.



- *Reuse of domain knowledge:* Domain knowledge is subject to change. But however fast the domain knowledge may change the underlying concept remains the same and is very valuable, history of any domain is the record of most valuable milestones and achievements. This info when represented as ontology will make it reusable, sharable and also safe.

- *Separating domain knowledge from operational knowledge:* The domain knowledge and the operational knowledge are definitely connected but they do not always go together. For instance, there are many varieties of software design but the operational procedure need not follow any of those design rules. Thus, the operational knowledge can exist independent to it. These differentiations can be made with ease when represented with task driven ontologies.



- *Analyzing domain knowledge:* Analysis and navigation though the terms of a specific domain is made simple when represented with an ontology. Ontology also defines the way these terms are related. Example: Computing Ontology where all the terms of computing domain are classified into areas and these areas have sub-classes that are bound together with relations. More about computing ontology can be found in Section 5.

3.3 Properties and Relations

We propose ontology to be, a formal representation of concepts with well defined relationships amongst these concepts. These concepts are source of shared terms that are understood and processed by machines. A typical ontology consists of a hierarchical structure of concepts and their relations of a particular Domain. An ontology is very similar to a taxonomy which is a hierarchical structure of information with only relation relationships amongst the terms being "is-a/has-a" they are easy to understand and very less expressive.

The is-a/has-a relationship is defined as a basic relation amongst the terms of a Taxonomy. It is contrasted by a *has-a* relation. Fig-3.3.1 describes taxonomy where

- Parent-1, Child-1 & Child-2 are related by an *is-a* or and *has-a* relation meaning the child Child-2 has a named Parent.
- The properties, concepts and other defined entities of parent can be inherited by the child but the vice versa is not true.
- Similarly, Grand Child-3 and Grand Child-4 are in relation with Child-2.

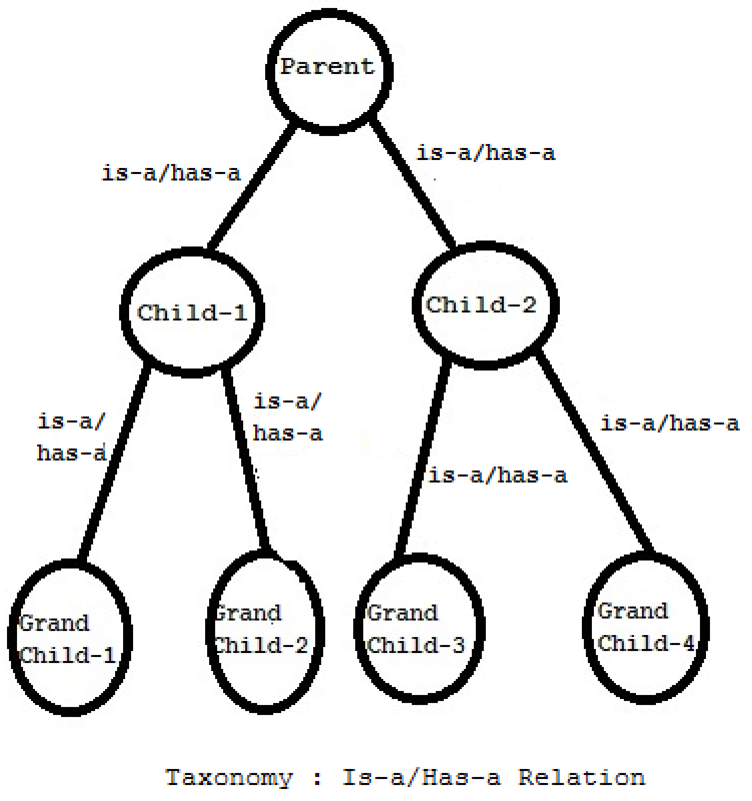


Fig-3.3.1 Taxonomy: is-a/Has-a relation

One interesting thing is the possibility to see a child having two mutually exclusive parents that is more than one parent. There are always questions open for thinking of possible relation or possible effects a node has by other nodes of its hierarchy, its parents, children, siblings, ancestors- refer to Fig-3.3.2.

In Fig-3.3.2 shows terms and relations at various levels of an ontology. It is not important the way these terms are ordered except for the fact that they have properties inherited from root. The relations defined in Fig-3.3.2 shows the way the terms/concepts are related, apart from a hierarchical is-a/has-a relation the terms are also connected with explicit relation such as belongs, derives, takes-from, used-by, similar-to which shows the way these resources are connected or related in the real world instances. Therefore we term ontology as a complex form of taxonomy having more terms, more relations and this is more expressive.

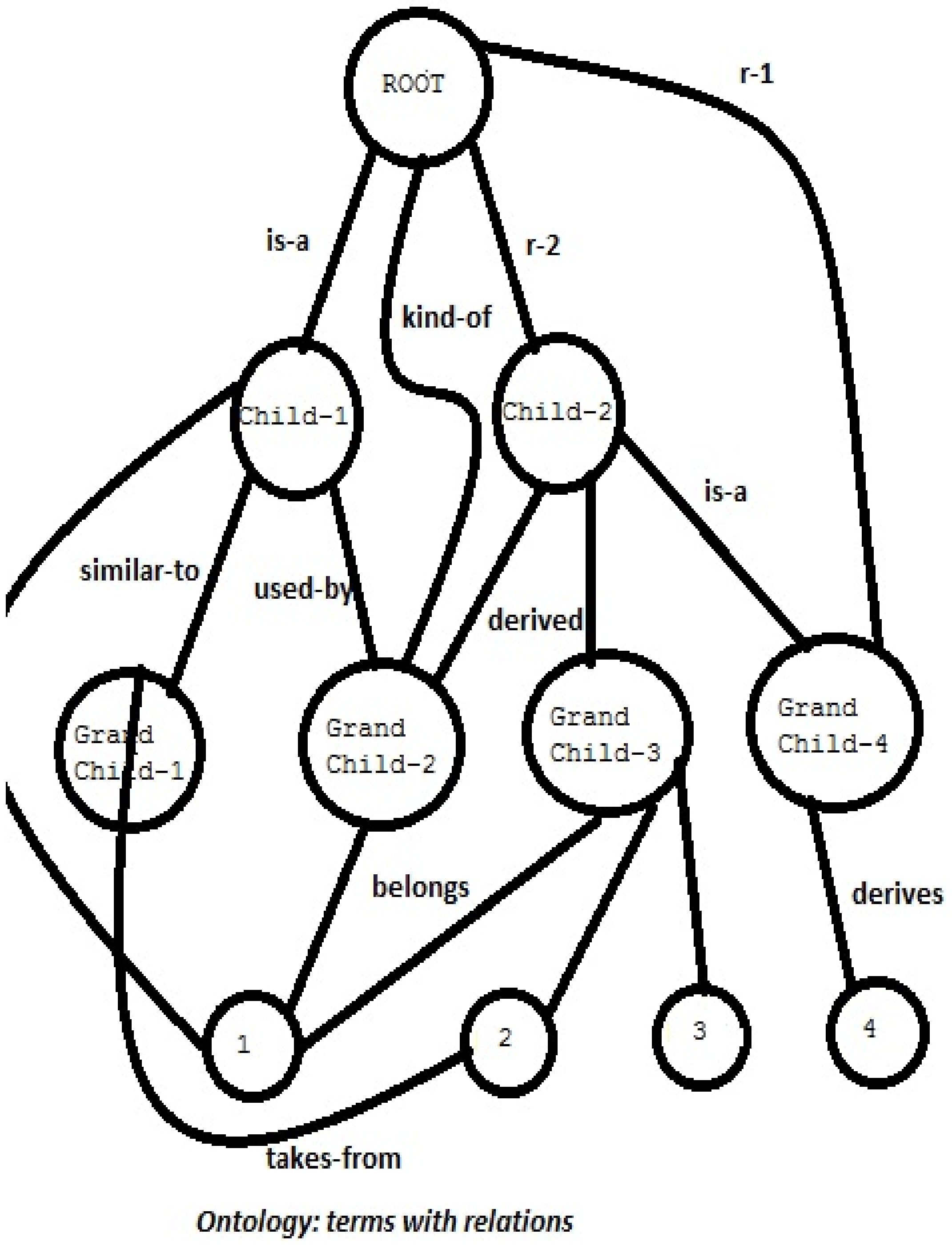
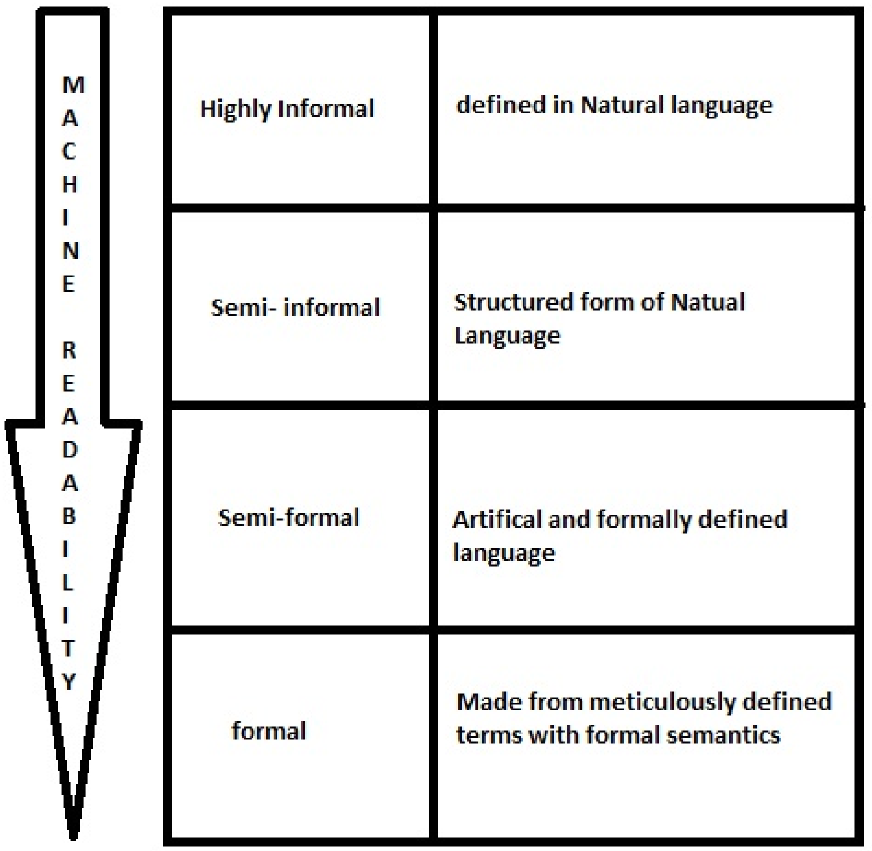


Fig-3.3.2 Ontology with terms and relations

3.4 Description

Ontologies can be highly informal if they are defined in natural language but they are not pure ontologies because they lose the machine readability. Similarly, we can say ontologies are semi informal, if they are created from a restricted or structured from of natural language. The other types are as shown:


MACHINE READABILITY

Highly Informal	defined in Natural language
Semi- informal	Structured form of Natual Language
Semi-formal	Artifical and formally defined language
formal	Made from meticulously defined terms with formal semantics

Fig-3.4.1 Formal Ontology

As we go down the table the machine readability increases.

The formal ontologies are those which are created using semantic web languages defined in Section-2. A highly informal ontology is never an ontology as it is not machine readable.

3.4.1 Types of ontologies

There are many factors that differentiate one kind of ontology from another. Different practitioners have different perspective toward their ontologies as shows in the Fig-3.4.1.1

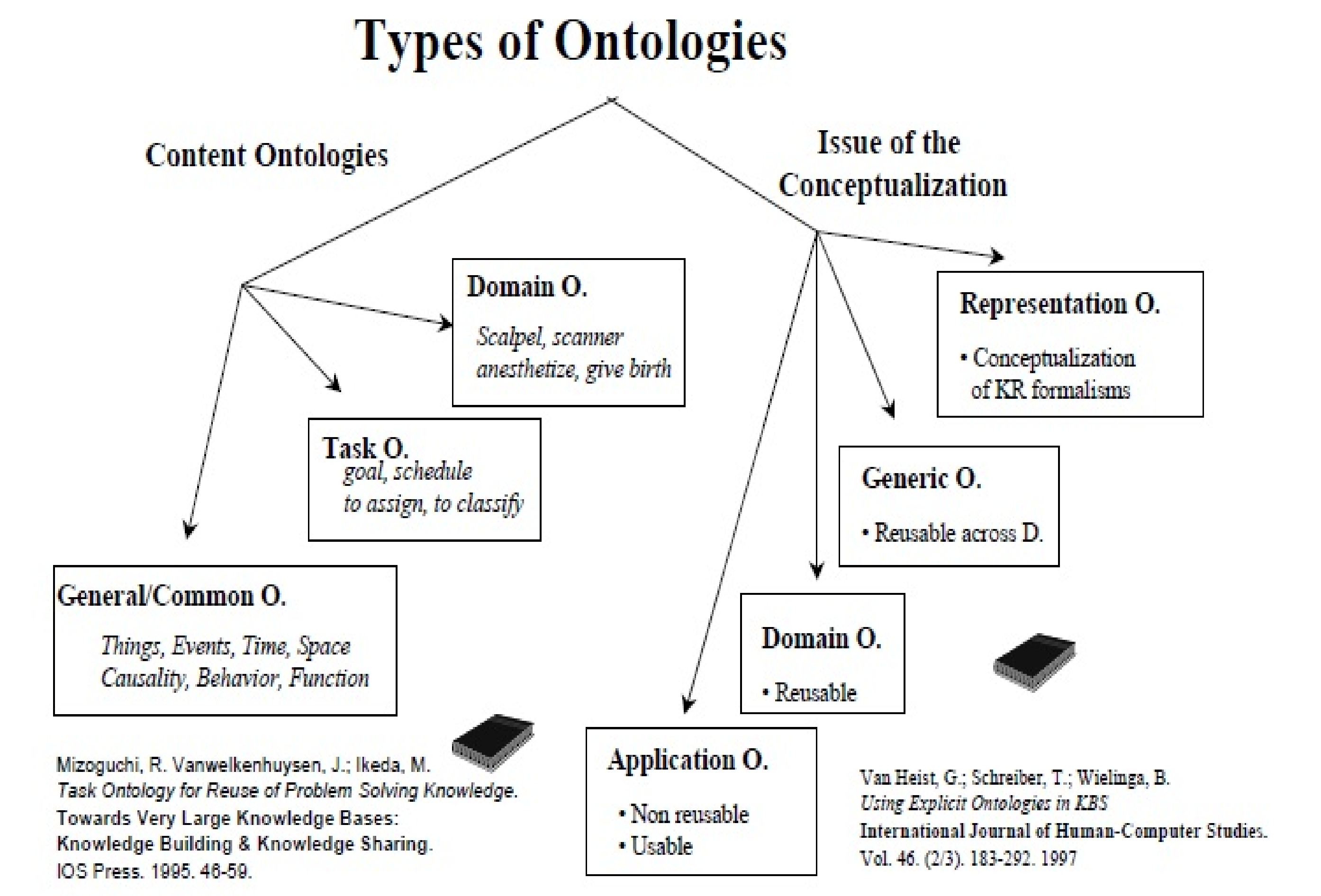

Types of Ontologies
Content Ontologies
Issue of the Conceptualization
Domain O.
Scalpel, scanner anesthetize, give birth
Task O.
goal, schedule to assign, to classify
General/Common O.
Things, Events, Time, Space Causality, Behavior, Function
Representation O.
• Conceptualization of KR formalisms
Generic O.
• Reusable across D.
Domain O.
• Reusable
Application O.
• Non reusable
• Usable
Mizoguchi, R. Vanwelkenhuysen, J.; Ikeda, M.
Task Ontology for Reuse of Problem Solving Knowledge.
Towards Very Large Knowledge Bases: Knowledge Building & Knowledge Sharing.
IOS Press. 1995. 46-59.
Van Heist, G.; Schreiber, T.; Wielinga, B.
Using Explicit Ontologies in KBS
International Journal of Human-Computer Studies.
Vol. 46. (2/3). 183-292. 1997

Fig-3.4.1.1 Type of ontologies [21]

We focus on certain aspects like scope, descriptiveness etc.

- *Level of Description*: Depending on the depth of the hierarchical structure and number of relations describing properties, an ontology could be a controlled vocabulary or a common ontology with terms and relations. A controlled vocabulary is a way of organizing knowledge for easy retrieval. The ontologies of this type are generally classified as Upper Ontologies which means the ontologies which are designed to describe general things. [22]

- *Conceptual Scope*: Ontologies also differ based on the purpose of their creation/use of their content in other words the necessity of the ontology. The popular distinction of this class is Domain ontology and Task ontologies. [22]

  - ✓ *Domain Ontologies:* are the ontologies designed to describe terms specific to a particular domain.
    Example: Computing Ontology.

  - ✓ *Task Ontologies:* are the ontologies are designed to describe a specific task. Example: Fixing a car.

3.4.2 Instantiation of Ontologies - Creation of TBox and ABox

This is very similar to schema described in a Databases or an XML. The schema has the structure of an ontology and the instances to that schema hold the actual data. In ontological terms TBox or the Terminology box has the concepts and the ABox or Assertion box has the facts associated to those concepts. [22]

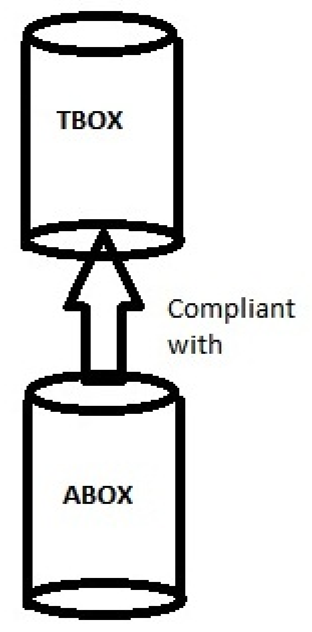


Fig-3.4.1.2 Knowledge Base [22]

These ontologies are generally application based ontologies which are developed for a specific application with known entities.

3.5 Summary

Ontology is specification of a concept. Every ontology knowingly or unknowingly fall into any type described in Fig-3.4.1.1 but the ontology surely has to be formally described with proper relations to enable machine readability.

4. Ontology Modeling - How

Ontology modeling mainly consists of

- Defining the concepts also called Classes.
- Arranging classes in proper hierarchy.
- Defining the properties - both simple and complex.
- Creating instances- objects associated to the classes.

The choice of modeling is dependent on the requirement, scope and complexity of the terms involved. For years now ontology development has been one of the most interesting topic of research, many scientists/developers have proposed a variety of methods of modeling ontologies which are introduced in section 4.1 [23]

4.1 Existing Methods

Many modeling techniques were proposed in the last decade. Some of them are described here.

Uschold and King (1995): has proposed an ontology development methodology that consists of four phases

1. *Purpose*: this is the phase to identify the purpose of the ontology and the scope of its application is analyzed.
2. *Building*: This is further divided into 3 categories
    a. *Capture*: the key concepts, relationships among these concepts are identified and are named according to the requirement.
    b. *Coding*: is the phase where the captured information is represented as a formal language. This phase includes choosing the feasible language.
    c. *Integrating*: is the final phase of to build the ontology where pre-existing ontologies are varied for merging.
3. *Evaluating and documentation*: where the ontology is checked to meet all the proposed requirements. Any unnecessary terms present in the ontology are trimmed and finally documented.

Gruninger and Fox(1995): Proposed an ontology development methodology as a part of the TOVE Enterprise Modeling project. According to this method, ontologies are created by answering a set of competency questions. These formal competency questions are answered and ontology commitments and terms involved are identified. Once the terms are identified the relationships among them are derived. The key idea of having the competency questions is to make the build process simpler by representing these questions as axioms for the defined terms.

Unified Model (1996): Uschold proposed an ontology building procedure which is the combination of the best phases of above two methodologies. This method is called the Unified method of ontology development. This involves the following stages:

1. *Purpose*: this step identified the purpose of the building ontology. Identification could be done by analyzing the ontology from a user perspective or by evaluation through use cases or by creating the competency questions which is close to a user requirement document.
2. In the second stage of development the developer decides the concepts, the terms and the relationship binding them.
3. These terms and properties are then structured using a preferred ontology editor.

Every phase in this method results in a document that describes the current state of the ontology.

Sugumaran and Storey(2002): proposed a heuristic based ontology development methodology where

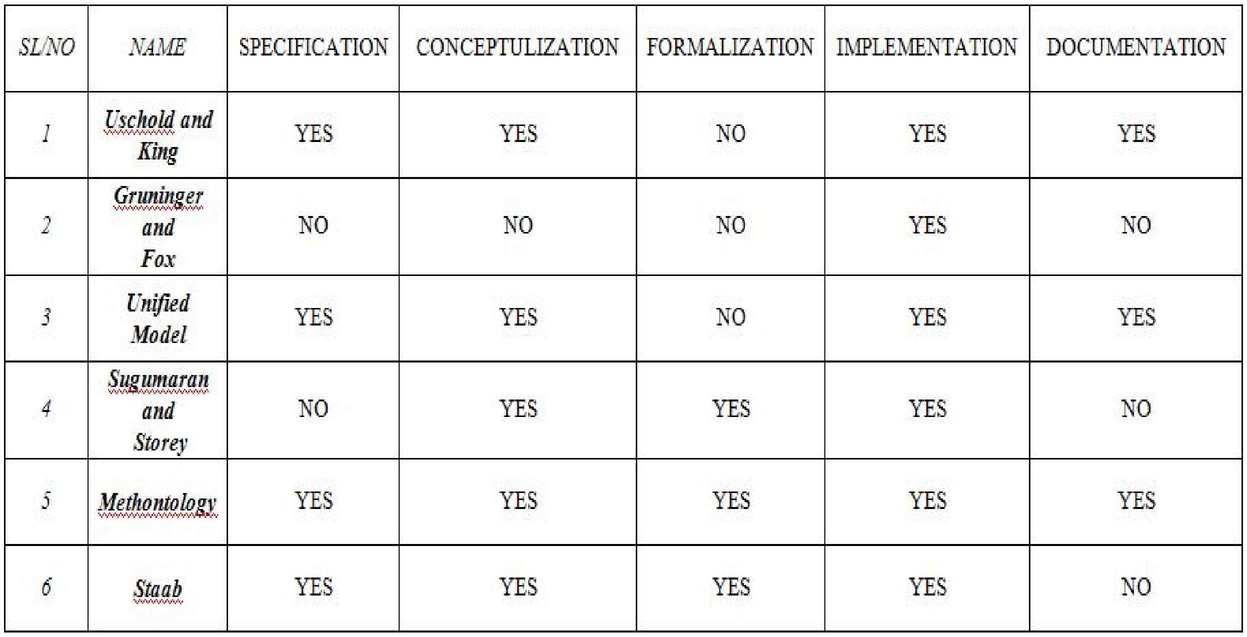
1. The terms are identified using use cases and these terms are compared with an online thesaurus to identify and trim off the synonyms.
2. The second phase is to identify the relationships among these terms. This method looks for three different kinds of relations called
    a. *Generalization*: generalization binds the terms into a hierarchy using an is-a/has-a relation.
    b. *Relation between ontologies*: where one or more ontologies are related.
    c. *Identifying the constraints*: where constraints such as the relationships amongst the terms.
3. The final step is identified the high level constraints such as the domain and domain dependencies.

Methontology(1997): This method is proposed by Fernandez to build the ontologies from the scratch. The following steps are involved in building.

1. *Purpose*: The phase where the purpose of the ontology is specified along with the level of formality and scope.
2. *Knowledge Acquisition*: In this phase all the relation knowledge is gathered by analyzing texts or available tools.
3. *Conceptualization*: In this step a glossary of terms is prepared and they are grouped according to the concepts and are later represented into tables or formulae.
4. *Implementation*: This phase involves looking for an already existing ontology that could be used and finally an ontology is coded in a formal language.
5. *Documentation*: Each and every phase is well documented. Thus, the final document gives all the information on the development.

Staab(2001): This method proposes a development life cycle to build an ontology. The following are the phases involved:

1. *Feasibility Study*: this is the phase where the domain of the ontology is identified and are is put into an organizational perspective.
2. *Ontology Kickoff*: this is the phase where a requirement specification document is prepared. This document consists of the design guidelines, knowledge sources, the users and desired application details of the ontology.
3. *Refinement*: this is the phase where the initial draft the ontology is revised and a formal ontology created with chosen language.
4. *Evaluation*: this is the phase where the ontology is compared with the requirements specification document and is tested against the target application.
5. *Maintenance and evolution*: phase where the ontology is updated new terms are inserted and unnecessary terms are deleted.



SL/NO	*NAME*	SPECIFICATION	CONCEPTULIZATION	FORMALIZATION	IMPLEMENTATION	DOCUMENTATION
1	***Uschold and King***	YES	YES	NO	YES	YES
2	***Gruninger and Fox***	NO	NO	NO	YES	NO
3	***Unified Model***	YES	YES	NO	YES	YES
4	***Sugumaran and Storey***	NO	YES	YES	YES	NO
5	***Methontology***	YES	YES	YES	YES	YES
6	***Staab***	YES	YES	YES	YES	NO

Table-4.1.1: Comparison of existing methods.

4.2 Modeling ontologies as software

Ontology can be built

- From the scratch
- By re-engineering the available ontologies
- By fusion or a merger of 2 or more ontologies.

From the structural perspective, ontology looks analogous to the many other streams of computing science such as an Object Oriented design concept, a Database system with well defined Entities and Relations.

4.2.1 Modeling ontology using Unified Modeling Language (UML)

The unified modeling language is one of the prominent ways of engineering software. UML could play an active role in bridging the gap between software engineering and ontological engineering.

- It can be used as a graphical front end for ontology modeling language.
- It can be handy in mapping the ontology terms and relations.
- There are tools that directly convert UML diagrams to Java Classes and RDF schemas from a class.

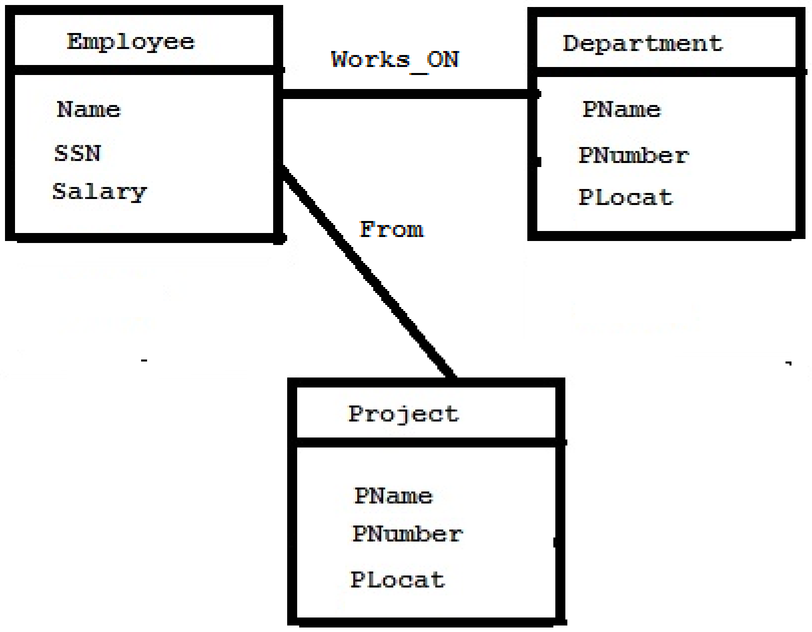


Fig-4.2.1.1 UML of an Object Model.

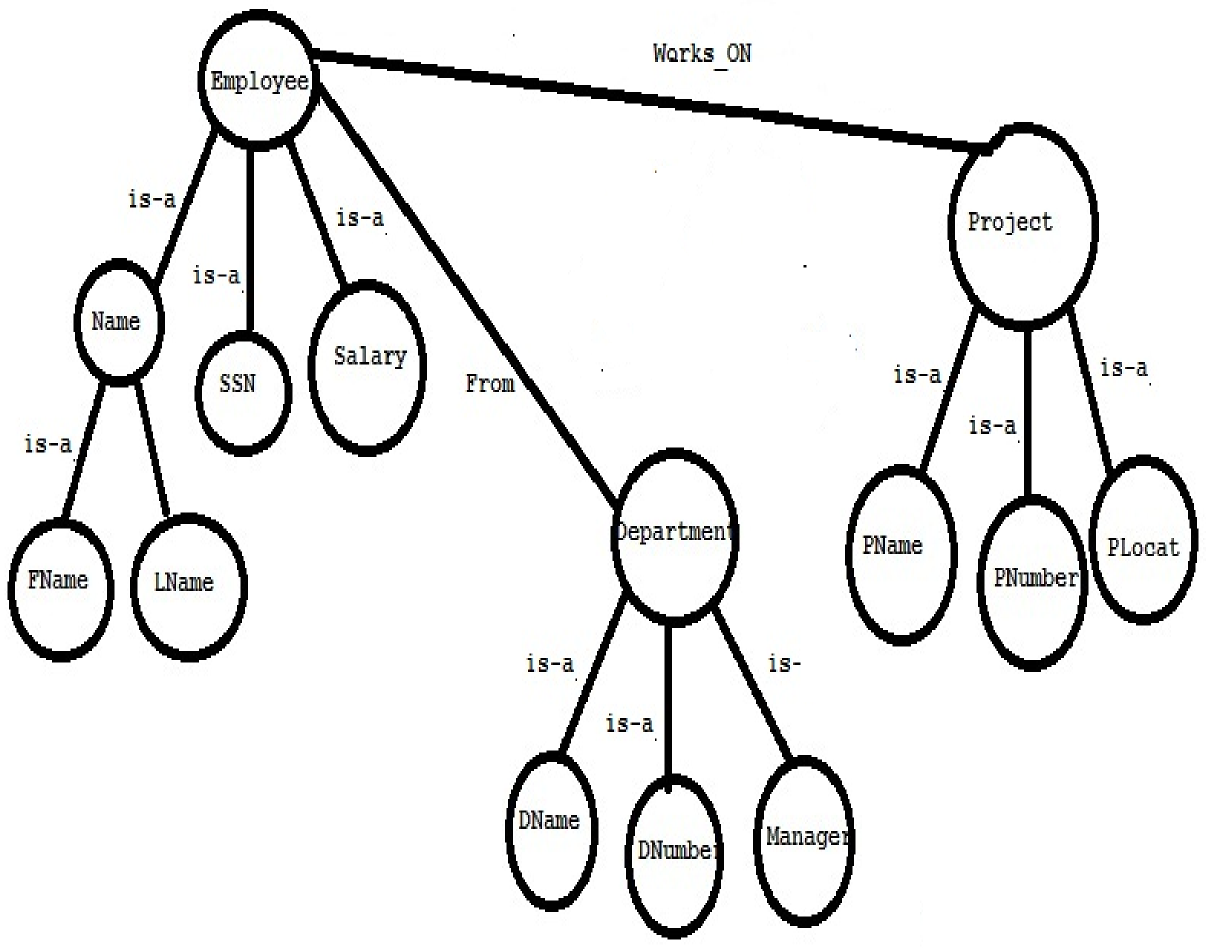


Fig-4.2.1.2 Ontology Model for above UML.

4.2.2 Modeling ontology using a Database

Ontology structure closely replicates a Database system. E-R Diagram could be helpful in ontology design, providing visual information on complexity of ontology with

- Classes being the ER-Entities
- Attributes being the ER-Attributes
- Ontology class relation are the ER-Entity relations
- Tertiary relations and other complex properties can also be representing at ease.

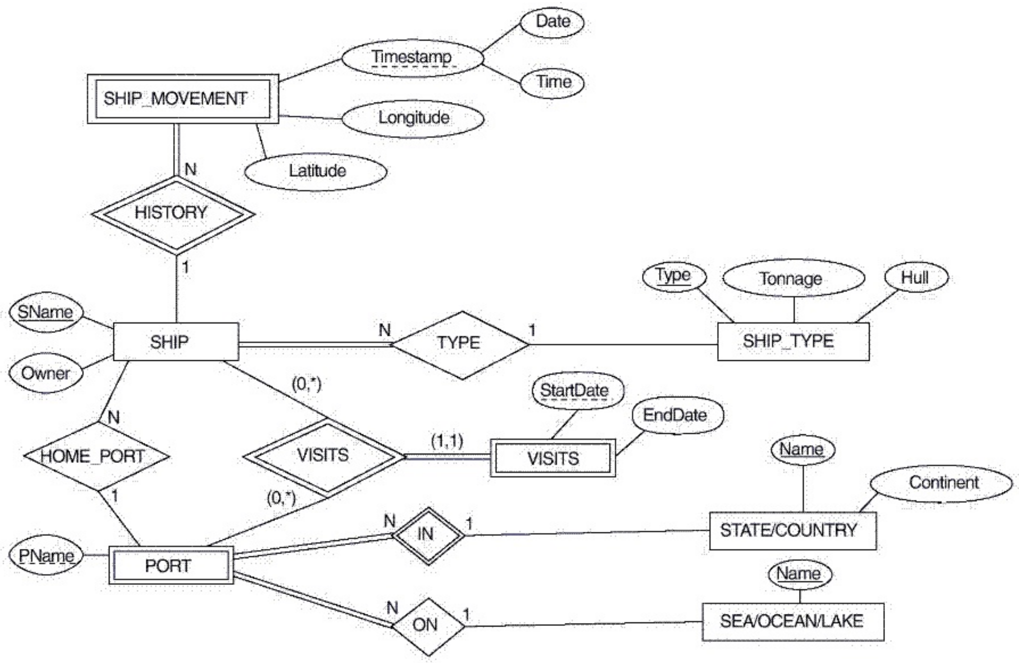


Fig-4.2.2.1 ER Diagram of a Database Schema

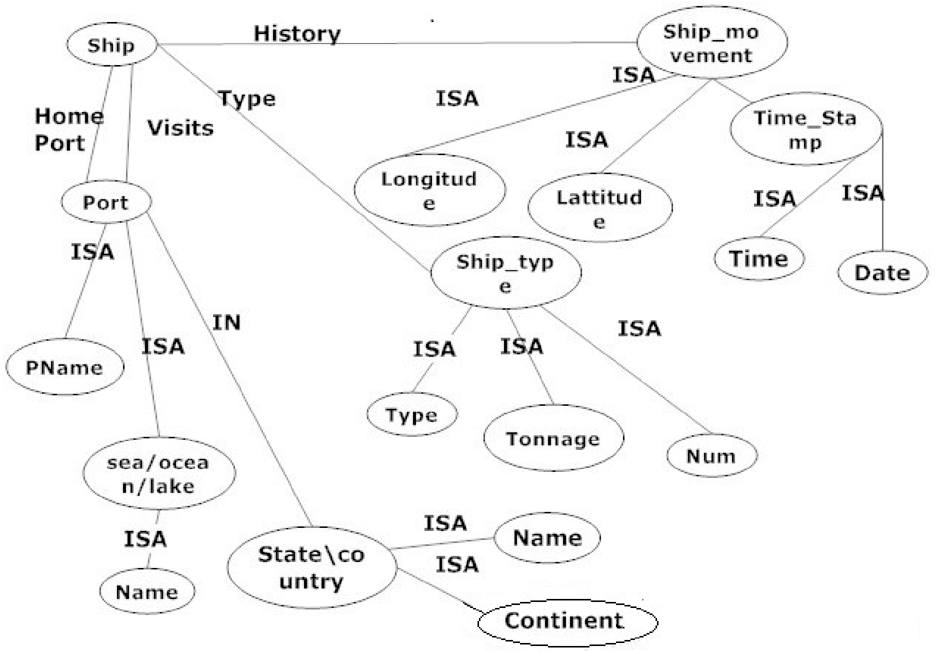


Fig-4.2.2.2 Ontology of a Database Schema

4.2.3 SDLC VS Modeling Languages

A software is a set of computer programs, procedure and possibly associated documentation and data pertaining to the operation of computer system. If ontologies are treated as software then all the methodologies of software development can be implemented for ontology design.

- *Activities in Software Development:*
    1. *Planning:* This is the phase for requirement analysis. The final end product is documented according to customer's perspective. This phase does not care about why the software is created for.
    2. *Implementation:* This phase is where the developers/engineers code the project.
    3. *Testing:* One of the most important phases of a software development which ensures the software does not break.
    4. *Documentation:* Is the phase where a written form of development phases helpful for software maintenance and future enhancements.
    5. *Deployment:* This phase is to make a release and distribution of the project to client or users.
    6. *Maintenance:* the phases which look at user or developer requests for enhancement and bug fixes.

All the methods presented in section 4.1 propose various ways of ontology modeling; some have proposed their own design model and other have done it by mere inspection and manual knowledge gathering. Fig- 4.2.1 shows a comparison of all these methods.

4.2.4 ODLC – Ontology Development Life Cycle

After carefully analyzing these methods and looking ontologies from a software engineering perspective we have come up with the following phase of ontology development called ODLC-Ontology Development Life Cycle.

- *Activities in an Ontology Development:*
    1. *Analysis:* This phase describes the purpose of the ontology which helps in identifying the domain of the ontology.
    2. *Knowledge gathering:* This phase is dependent on the domain of the ontology identified in the analysis phase. This domain influences the scope of the terms being gathered in this phase.
    3. *Enumeration and Refinement:* Enumeration is the phase where the gathered terms are grouped/ tabulated according to their closest relations. This is manually done with human understanding of the domain and these tables are trimmed by domain experts.

4. *Building:* The phase where the actual ontology is built. This can be done using hard coded programming or any ontology editors. *(more information on ontology editors is in Section- 4.4)*
5. *Enhancement:* Knowledge in every domain keeps on changing which demands for increasing of concepts/terms in the ontology.
6. *Documentation:* A well structured document that keeps track of build process.
7. *Maintenance:* Maintenance is the phase were the ontology is periodically updated and maintained.

ODLC is an extension of Methontology with the enhancement and maintenance phases added to the build process.

4.3 Languages to model ontologies - More OWL

As described in section 2.3.4, OWL is one of the finest languages to describe an ontology. It is further categorized into 3 sub-languages depending on their strength of expressiveness.

4.3.1 OWL – Lite

OWL-Lite is the least expressive sublanguage and is syntactically simple. Its applications are to create simple hierarchies with simple constraints among the terms.

Properties or Class Expressions for owl-Lite:

- Conjuction: This is a property with restricted range. This is permissible only to class identifiers.
- Property Restrictions: this property amongst the elements that share a property restriction such as a *value restriction* or an *existential restriction*.
- Number Restrictions: this property imposes a cardinality restriction on the elements. OWL lite allows to use only 0 or 1 cardinality

```
<owl:Class rdfID=”Fruit”>
     <owl:interscetionOf rdf:parseType=”Collection”>
</owl:Class>
```

Example-4.3.1.1 Showing OWL-Lite

Further explanation on restriction is provided on section 4.4.1

4.3.2 OWL – DL

The expressiveness of OWL-DL falls in between OWL-Lite and OWL-Full. Description Logics(DL) are used to create and bind terms together. This language is generally used in creating classification hierarchies. Some restrictions such as a class CANNOT be an individual comes along with the language construct.

Properties allows in OWL-DL:

- Conjunction and disjunction – owl: intersectionOf and owl:unionOf respectively.
- Collection of individuals – owl:one of
- Number Restrictions: all positive integer values including 0.

```
<owl:Class rdf:ID ="#Fruits">
     <owl:Restriction owl:cardibality="1">
          <owl:onProperty rdf:resource="#fruitName"/>
     <owl:Restriction/>
</owl:Class>
```

Example-4.3.2.1 Showing OWL-DL

4.3.3 OWL – Full

OWL-Full is the most expressive sub-language of OWL. This is convenient to create automatic and computational hierarchy of terms. OWL Full could be said as a super set of both OWL Lite and OWL-DL.

```
<owl:Class rdfID="Fruit">
  <owl:interscetionOf rdf:parseType="Collection">
    <owl:Class rdf:ID ="#Fruits">
       <owl:Restriction owl:cardibality="1">
          <owl:onProperty rdf:resource="#fruitName"/>
          <owl:allValuesFrom rdf:resource="some resource"/>
       </owl:Restriction>
       <owl:Restriction>
          <owl:hasValue rdf:datatype="some datatype:String">
          </owl:hasValue>
       </owl:Restriction>
   <owl:interscetionOf>
</owl:Class>
```

Example-4.3.3.1 Showing OWL-Full

The following relations holds good at all times:

- Every legal OWL Lite ontology is a legal OWL DL ontology.
- Every legal OWL DL ontology is a legal OWL Full ontology.
- Every valid OWL Lite conclusion is a valid OWL DL conclusion.
- Every valid OWL DL conclusion is a valid OWL Full conclusion. [24]

4.4 Components of OWL Ontologies

An OWL ontology consists of Classes, Properties, and individuals.

Individuals: are the objects of the domain. In Example-4.4.1 shows that Red apples and Green apples are two different instances of same classes called Types_of_Apples.

```
        <Types_of_Apples rdf:ID="RED"/>

         <Types_of_Apples rdf:ID="GREEN"/>
```

Example- 4.4.1: Showing Individuals

Properties: are the binary relations amongst individuals.

- For example a property like hasSibling can be used to link two individuals like Jack and Jill.
- A property can also be inverse that is a property names *Uses* can have a inverse property called *UsedBy.*
- A property can be functional, that is a property having a single value which could also be transitive or symmetric.

```
<owl:ObjectProperty rdf:about="#IsPartOf">
   <owl:inverseOf>
     <owl:ObjectProperty rdf:about="#HasPart"/>
   </owl:inverseOf>
   <rdf:type rdf:resource="http://www.w3.orr/owl#TransitiveProperty"/>
</owl:ObjectProperty>
```

Example-4.4.2: Showing Transitive Property

Example-4.4.2 shows that the object property *IsPartOf* has a transitive property associated with it called *HasPart*. This relates two objects from both sides.

Classes: Classes are sets or collection of individuals. They are formally described with a Domain and Range to hold appropriate individuals. All classes are arranged in a super class hierarchy closely resembling to taxonomy.

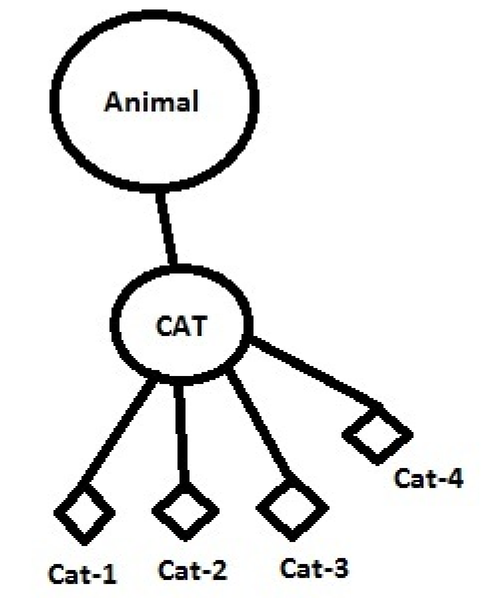


Fig-4.4.1 Super Class and its individuals

Statement 1: Animal is a Super-Class of Cat implies that all cats are animals.

Statement 2: All members of class Cat are also members of class Animal implies that any member inside Cat is and animal.

Closely looking at the above statements, statement 1 and statement 2 mean the same the difference is statement is machine readable while statement 2 is in natural language. There for expressing natural language statements with super-class and sub-class relationships can prove helpful for machine understandability.

4.4.1 Class Descriptions [25]

OWL descriptors are the basic building blocks of OWL classes. Class descriptors describe an owl class entity by class name or by specifying the class extension from another named or unnamed class. There are two predefined owl classes called

- owl:Thing is super class for every owl class that is every class that exists in owl is a subclass of OWL:Thing.
- owl:Nothing is subclass of every class, that is owl:Nothing is an empty set.

OWL has 6 different types of descriptors:

1. ***Class Identifier***: this describes a class through a class name.

`<owl:Class rdf:ID="Cat"/>` describes a class called Cat. The presence of `rdf:ID` here is because OWL is full extension of RDF.

2. ***Enumeration of individuals or instances of a class****:* This is a descriptor that describes an anonymous class using constraints as class extension. A class description of enumeration kind is defined using *owl:oneOf* property as shown in Example-4.4.1.1

```
            <owl:Class>
                <owl:oneOf rdf:parseType="Collection">
                     <owl:Thing rdf:about="#Apples"/>
                     <owl:Thing rdf:about="#Oranges"/>
                     <owl:Thing rdf:about="#Bananas"/>
                     <owl:Thing rdf:about="#Pineapples"/>
                      <owl:Thing rdf:about="#Mango"/>
                      <owl:Thing rdf:about="#Strawberries"/>
                      <owl:Thing rdf:about="#Blueberries"/>
                      <owl:Thing rdf:about="#Grapes."/>
                  </owl:oneOf>
          </owl:Class>
```

Example-4.4.1.1: Showing oneOf property for individuals

Example-4.4.1.1 shows that the one of the above mentioned fruits could be an individual to the class.

3. ***Property restriction***: is an anonymous class descriptor that contains set of all individuals that satisfy a particular restriction placed on them.

OWL provided two different kinds of property restrictions

- *Value constraint:* this puts constraint on the range of the property.
    - owl:allValuesFrom
    - owl:someValuesFrom
    - owl:hasValue

```
<owl:Class rdf:ID="SerialNumber">
   <owl:Restriction>
      <owl:allValuesFrom rdf:resource="Integers"/>
      <owl:allValuesFrom rdf:resource="Characters"/>
       <owl:hasValue rdf:datatype="alphaNumeric" />
  </owl:Restriction>
</owl:Class>
```

Example-4.4.1.2: Showing Property Restriction

Example 4.4.1.2 describes a class representing a serial number which could be alphabetical or numeric. Therefore, it can take all the values from Integers or all values from characters. In order to restrict the combination we need to specify the cardinality constraint.

- *Cardinality constraint:* this restrict on number of values a property can take.
    - owl:maxCardinality
    - owl:minCardinality
    - owl:cardinality

The lists above show some property restrictions available in owl.

```
<owl:Class rdf:ID="SerialNumber">
   <owl:Restriction owl:mincardinality="2">
      <owl:allValuesFrom rdf:resource="Integers"/>
      <owl:allValuesFrom rdf:resource="Characters"/>
      <owl:hasValue rdf:datatype="alphaNumeric" />
   </owl:Restriction>
</owl:Class>
```

Example-4.4.1.3: Showing Cardinality Constraint

Example 4.4.1.3 restricts saying the serialNumber must have atleast two integers and two characters.

4. ***Intersection of two or more classes***: This describes intersection amongst multiple classes using *owl:intersectionOf* property. In other words it describes a class to have set of individuals that belong to a class extension and list of classes mentioned in the description.

```
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class>
                  <owl:oneOf rdf:parseType="Collection">
                    <owl:Thing rdf:about="#fruits" />
                    <owl:Thing rdf:about="#vegetables" />
                  </owl:oneOf>
                </owl:Class>
                <owl:Class>
                  <owl:oneOf rdf:parseType="Collection">
                    <owl:Thing rdf:about="#wine" />
                    <owl:Thing rdf:about="#vodka" />
                  </owl:oneOf>
               </owl:Class>
               <owl:Class>
                  <owl:oneOf rdf:parseType="Collection">
                    <owl:Thing rdf:about="#Novel" />
                    <owl:Thing rdf:about="#Text" />
                  </owl:oneOf>
               </owl:Class>
            </owl:intersectionOf>
       </owl:Class>
```

Example-4.4.1.4: InterscetionOf property

Example 4.4.1.4 is more like doing an *and* operation among the classes. This example is the descriptor of a class which allows choosing one from each available sub-collection classes forming a parent collection. That is one from either fruits or vegetable plus one from either wine or vodka and one from either Novel or Text book.

5. ***Union of two or more classes**:* This describes intersection amongst multiple classes using *owl:unionOf* property. In other words it describes a class to have set of individuals that belong to at least one of either the class extension or list of classes mentioned in the description.

```
<owl:Class>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Fruits" />
        <owl:Thing rdf:about="#vegetables" />
      </owl:oneOf>
    </owl:Class>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#wine" />
        <owl:Thing rdf:about="#vodka" />
      </owl:oneOf>
    </owl:Class>
  </owl:unionOf>
</owl:Class>

Example-4.4.1.5: unionOf property
```

This descriptor is more like doing an OR operation on the owl classes.

6. ***Complement of two or more classes**:* This describes intersection amongst multiple classes using *owl:complementOf* property. In other words it describes the class extension consists of those individuals that are NOT members of the class extension of the complement class.

```
<owl:Class>
  <owl:complementOf>
    <owl:Class rdf:about="#Meat"/>
  </owl:complementOf>
</owl:Class>
```

This descriptor is more like doing and NOT operation on the owl classes.

4.4.2 Class Axioms [25]

The class descriptors discussed in section 4.4.1 are used to build a class through class axioms. A class axiom provides information of a class. Information such as the characteristics and nature of a class can be identified from class axioms like

rdfs:subClassOf: This describes the extension of a class or how two classes are related in a hierarchy.

```
<owl:Class rdf:ID="apple">
      <rdfs:subClassOf rdf:resource="#fruit" />
</owl:Class>
```

Exmaple-4.4.2.1: Showing subClassOf

Axiom in Example-4.4.2.1 declares a subclass relation between two classes. A class could get more complex when classes are nested with subClassOf relations as shown in Example-4.4.3.1

```
 <owl:Class rdf:ID="FruitSalad">
  <rdfs:subClassOf rdf:resource="#Apple"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
rdf:resource="#hasFruitType"/>
      <owl:someValuesFrom>
        <owl:Class>
          <owl:oneOf rdf:parseType="Collection">
            <owl:Thing rdf:about="#Peaches"/>
            <owl:Thing rdf:about="#Pineapple"/>
          </owl:oneOf>
        </owl:Class>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Example-4.4.2.2 showing complex subclassOf hierarchy *owl:equivalentClass:* Describes the equality among two classes meaning one class has same extension as the other class.

```
<owl:Class rdf:about="#Apple">
     <equivalentClass rdf:resource="#KingOffruits"/>
</owl:Class>

 Example-4.4.2.3: Showing Equivalent Class property
```

Equivalent class does not mean that the classes are same as other, King of fruits is the concept which is related to Apple but not equal to apple. If two concepts or class are to be said equal then the relation sameAs is used.

owl:sameAs saying that two classes are exactly same.

owl:disjointWith: describes that two calsses have zero members in common.

```
<owl:Class rdf:about="#Man">
     <owl:disjointWith rdf:resource="#Woman"/>
</owl:Class>

     Example-4.4.2.4: Showing disjointWith property
```

4.4.3 OWL Properties

OWL has two types of properties defined to describe relations among the terms.

- *Object Properties:* These properties link individuals together. OWL defines object properties using built in property class called owl:ObjectProperty, that is any object property defined in owl is an instance of the owl:ObjectProperty class.

```
     <owl:ObjectProperty rdf:ID="hasSibling" />

Example-4.4.3.1: showing hasSibling property
```

Example-4.4.3.1 shows an object property that relates individuals with a property called *hasSibling.*

- *Datatype Properties:* The data type properties links individuals with other values. These are defined using owl:DatatypeProperty class.

These two property classes are again instances of parent class called `rdf:Property.`

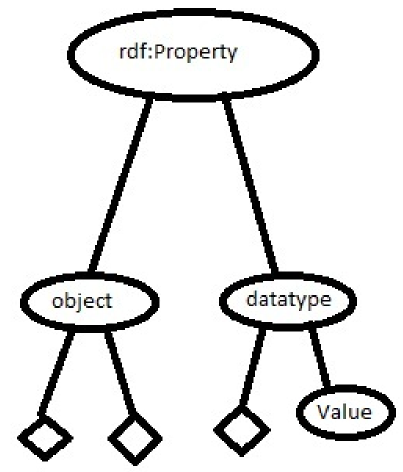


Fig-4.4.3.1- hierarchy of rdf:property

Along with all the above explained properties owl also supports the following constructs.

1.	**RDF Constructs**

	`rdfs:subPropertyOf`
	`rdfs:domain`
	`rdfs:range`
	`rdfs:comment`
	`rdfs:label`
	`rdf:isDefinedBy`
	`Class`
	`Individual`

2.	**Relation to Realtions**

	`owl:inverseOf`
	`owl:equivalentProperty`
	`owl:FunctionalProperty`
	`owl:InverseFunctionalProperty`
	`owl:ObjectProperty`
	`owl:DatatypeProperty`

3.	**Logical characteristics of Properties**
	`owl:TransitiveProperty` `owl:SymmetricProperty`
4.	**Individual Identities**

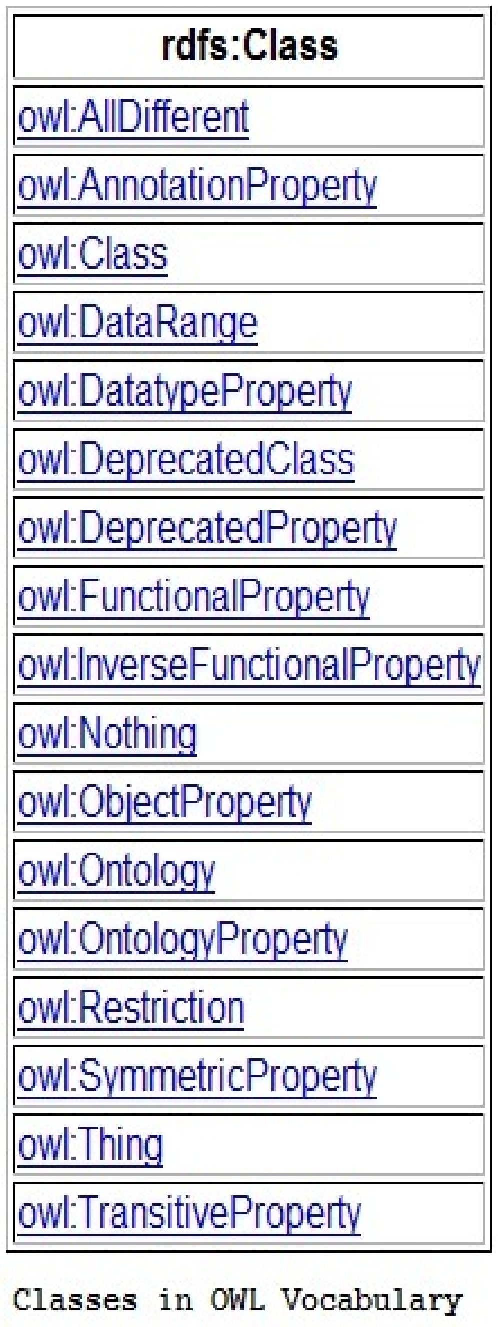
	`owl:sameAs`
	`owl:differentFrom`
	`owl:AllDifferent`

5.	**Cardinality Constraints**

	`maxCardinality`
	`minCardinaltiy`
	`Cardinality`

Table- 4.4.3.1: OWL Constructs

OWL provides the following built in classes:

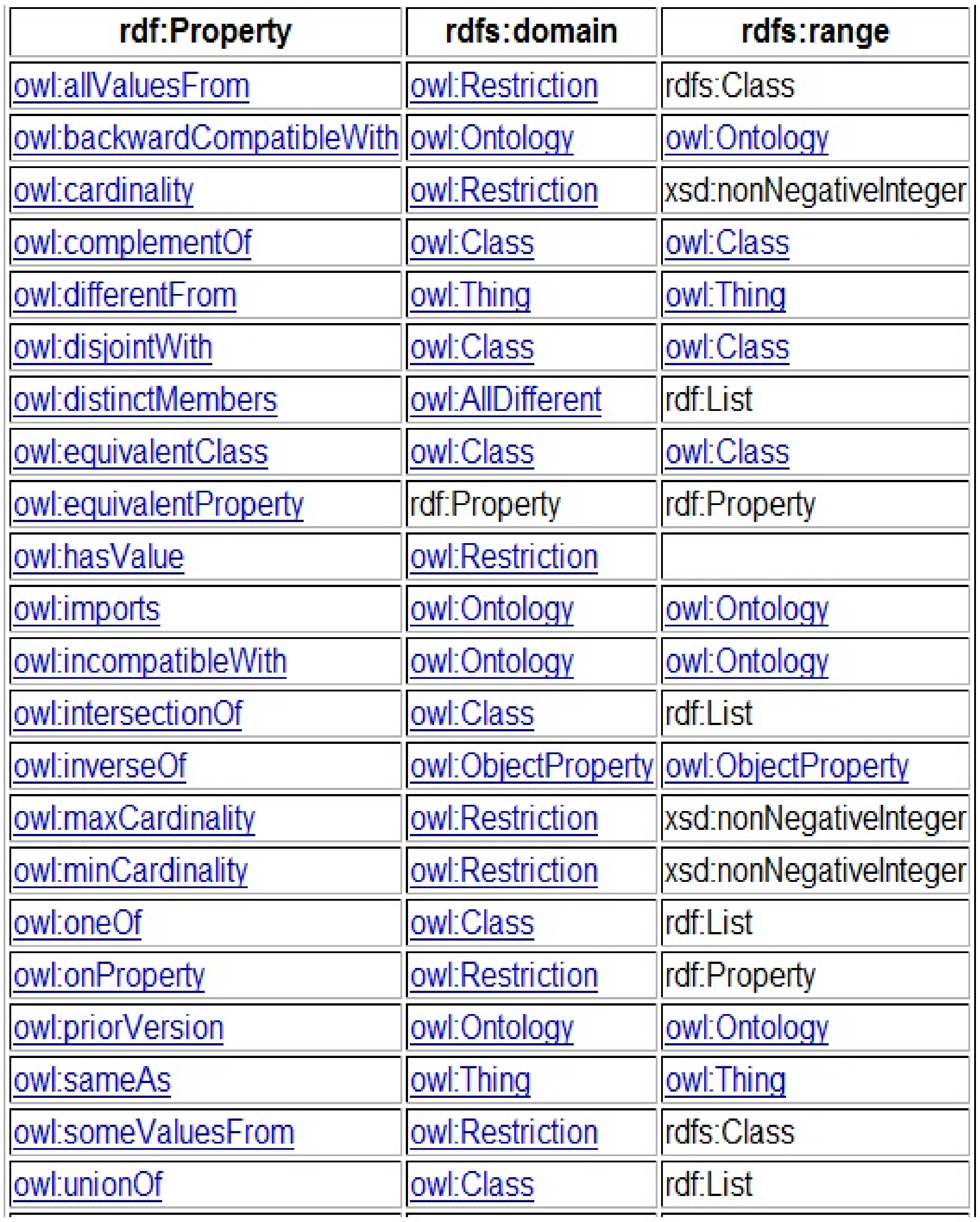


rdfs:Class
owl:AllDifferent
owl:AnnotationProperty
owl:Class
owl:DataRange
owl:DatatypeProperty
owl:DeprecatedClass
owl:DeprecatedProperty
owl:FunctionalProperty
owl:InverseFunctionalProperty
owl:Nothing
owl:ObjectProperty
owl:Ontology
owl:OntologyProperty
owl:Restriction
owl:SymmetricProperty
owl:Thing
owl:TransitiveProperty

Classes in OWL Vocabulary

Table-4.4.3.2 Built in Classes [25]

OWL provides the following built-in properties



rdf:Property	rdfs:domain	rdfs:range
owl:allValuesFrom	owl:Restriction	rdfs:Class
owl:backwardCompatibleWith	owl:Ontology	owl:Ontology
owl:cardinality	owl:Restriction	xsd:nonNegativeInteger
owl:complementOf	owl:Class	owl:Class
owl:differentFrom	owl:Thing	owl:Thing
owl:disjointWith	owl:Class	owl:Class
owl:distinctMembers	owl:AllDifferent	rdf:List
owl:equivalentClass	owl:Class	owl:Class
owl:equivalentProperty	rdf:Property	rdf:Property
owl:hasValue	owl:Restriction	
owl:imports	owl:Ontology	owl:Ontology
owl:incompatibleWith	owl:Ontology	owl:Ontology
owl:intersectionOf	owl:Class	rdf:List
owl:inverseOf	owl:ObjectProperty	owl:ObjectProperty
owl:maxCardinality	owl:Restriction	xsd:nonNegativeInteger
owl:minCardinality	owl:Restriction	xsd:nonNegativeInteger
owl:oneOf	owl:Class	rdf:List
owl:onProperty	owl:Restriction	rdf:Property
owl:priorVersion	owl:Ontology	owl:Ontology
owl:sameAs	owl:Thing	owl:Thing
owl:someValuesFrom	owl:Restriction	rdfs:Class
owl:unionOf	owl:Class	rdf:List

Properties on OWL Vocublary

Table-4.4.3.3 Built in Properties [25]

4.5 Tools to create ontology

Section 4.2 describes way to model ontology which is highly complex and time consuming. To make the ontology tasks easy many ontology modeling tools were created.

Ontology tools are different types depending on the task:

i. Ontology design tools
ii. Ontology merging tools
iii. Ontology evolution tools
iv. Ontology visualizing tools
v. Ontology querying tools and many others depending on the purpose.

4.5.1 Protégé [26]

Protégé is a multi plug-in open source ontology editor. It is a java based IDE allowing user to design, visualize, and merge ontology. It is developed at Stanford University to design ontologies using *protégé frames* and *protégé OWL.*

Protégé Frame: provides user with ability to create frame based ontologies that represents domain concepts with a set of classes to describe the properties and relations. These classes can further have instances associated to them.

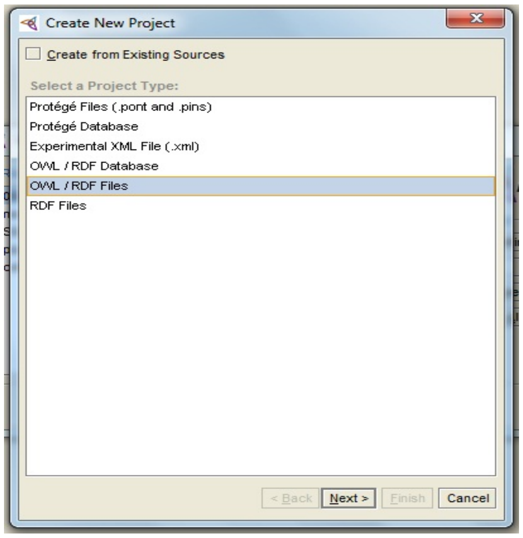
Protégé OWL: allows user to create ontologies for Semantic Web using OWL (Section-4.3).

Protégé being a GUI tool gives strength to interact and navigate through OWL structure. It also provides ability to convert owl files to other formats such as xml, protégé database, protégé project format and so on.

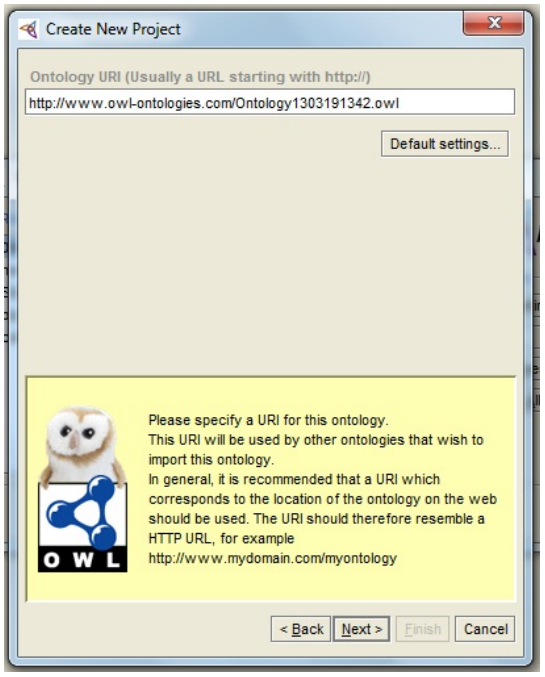
4.5.2 Walk through

A brief walk though of ontology creation is show in this section

- Once protégé is installed and a new project of type OWL/RDF is selected.



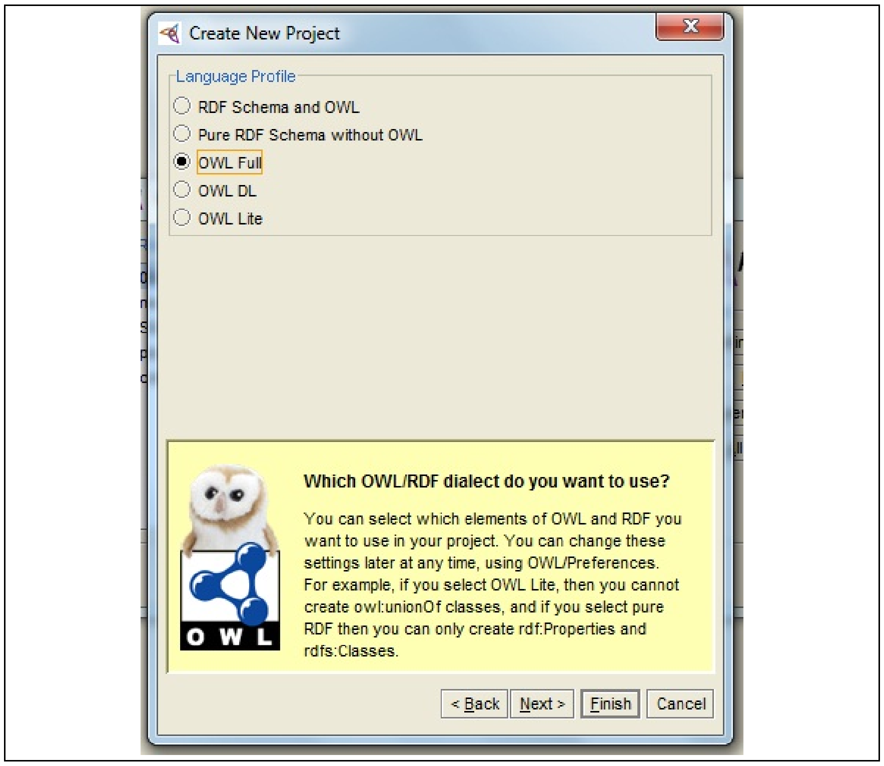
- A URL has to be chosen which acts as an identifier for the ontology.



http://www.owl-ontologies.com/Ontology1303191342.owl

Please specify a URI for this ontology.
This URI will be used by other ontologies that wish to import this ontology.
In general, it is recommended that a URI which corresponds to the location of the ontology on the web should be used. The URI should therefore resemble a HTTP URL, for example
http://www.mydomain.com/myontology

1. A language of choice to be selected depends on the factors such as level of expressiveness. For experimental purposes we choose OWL-Full as ontology language.



Create New Project

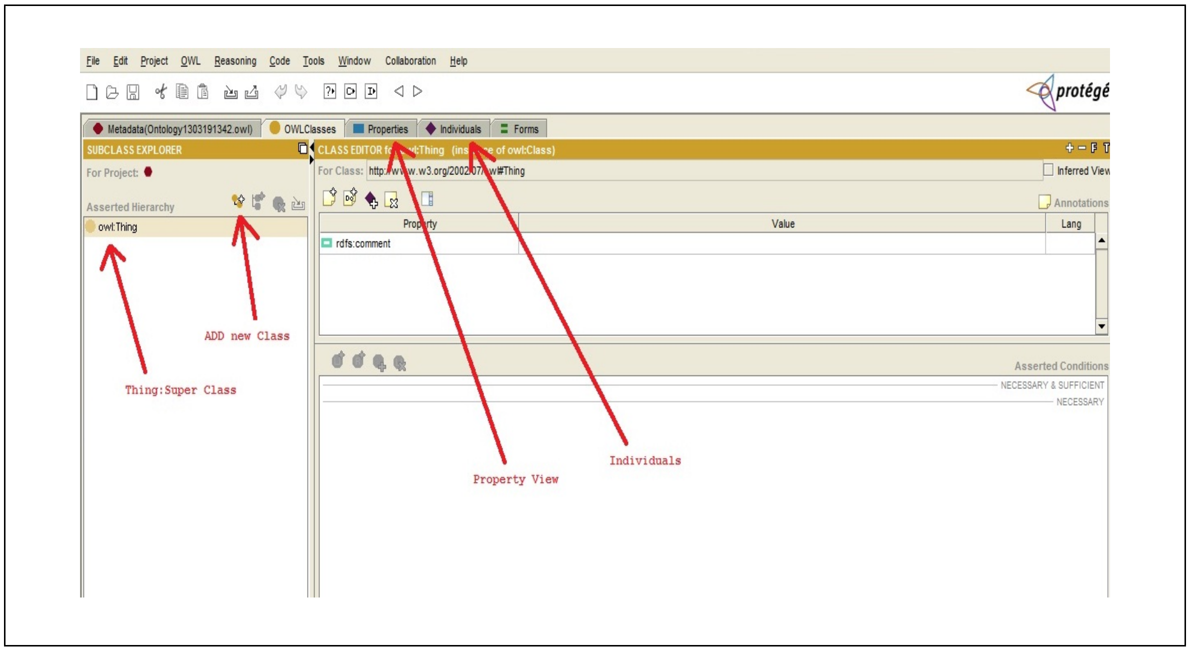
Language Profile

( ) RDF Schema and OWL
( ) Pure RDF Schema without OWL
[✓ OWL Full]
( ) OWL DL
( ) OWL Lite

Which OWL/RDF dialect do you want to use?

You can select which elements of OWL and RDF you want to use in your project. You can change these settings later at any time, using OWL/Preferences. For example, if you select OWL Lite, then you cannot create owl:unionOf classes, and if you select pure RDF then you can only create rdf:Properties and rdfs:Classes.

2. Either of the logic or property view can be selected which are interchangeable at any point of time. Logic view enables users to create class hierarchy and property view provides ability to assign properties or relationships among the classes.



4.6 Summary

ODLC provides a software engineering view of ontologies and defines an effective way of designing and maintaining ontology. Protégé is well known among the finest tools to design OWL Ontologies.

5. Computing Ontology

The Computing Ontology describes various disciplines, topics, and subtopics that belong to the domain of Computing Sciences. It also describes the ways in which these topics and subtopics are related to each other and to various sub disciplines. All the fields and subfields are organized as classes and objects in a hierarchical conceptual structure with parent, child or sibling relationships among them. Apart from these relations the fields/classes also have properties like Is a, Uses, Used By, Equals, Is Part of, Has Part etc.

ODLC (Ontology Development Life Cycle) is used to design new computing ontology.

5.1 Purpose

The ultimate goal of the Computing Ontology is to build and validate ontology of computing domain. A body of knowledge which allow to learn the overlaps among areas of computing domain.

All the concepts/terms in computing domain are represented as concepts and tied together with well defined relationships; such well defined ontology has the following applications:

- *Curriculum Recommendations*: Identifying the overlaps among the topics provides an ability to dig deep into areas allowing universities/colleges/institutions to identify the hierarchy of the course structure, required prerequisites for any topic.

- *Ontology Visualization*: A graphical user interface that facilitates free navigation amongst the terms to visual identifies the bridges between concepts.

- *Indexing Collections*: At this time ACM Computing Classification 1998 version is being used as controlled vocabulary for groups like CITIDEL, with the availability of owl file computing ontology can be used as a controlled vocabulary to index these collections and to maintain the digital library of computing materials.

- *Searching Content*: Information tagged with owl is precise and restricted, thus the search on such data would result in accurate search results.

5.2 Description

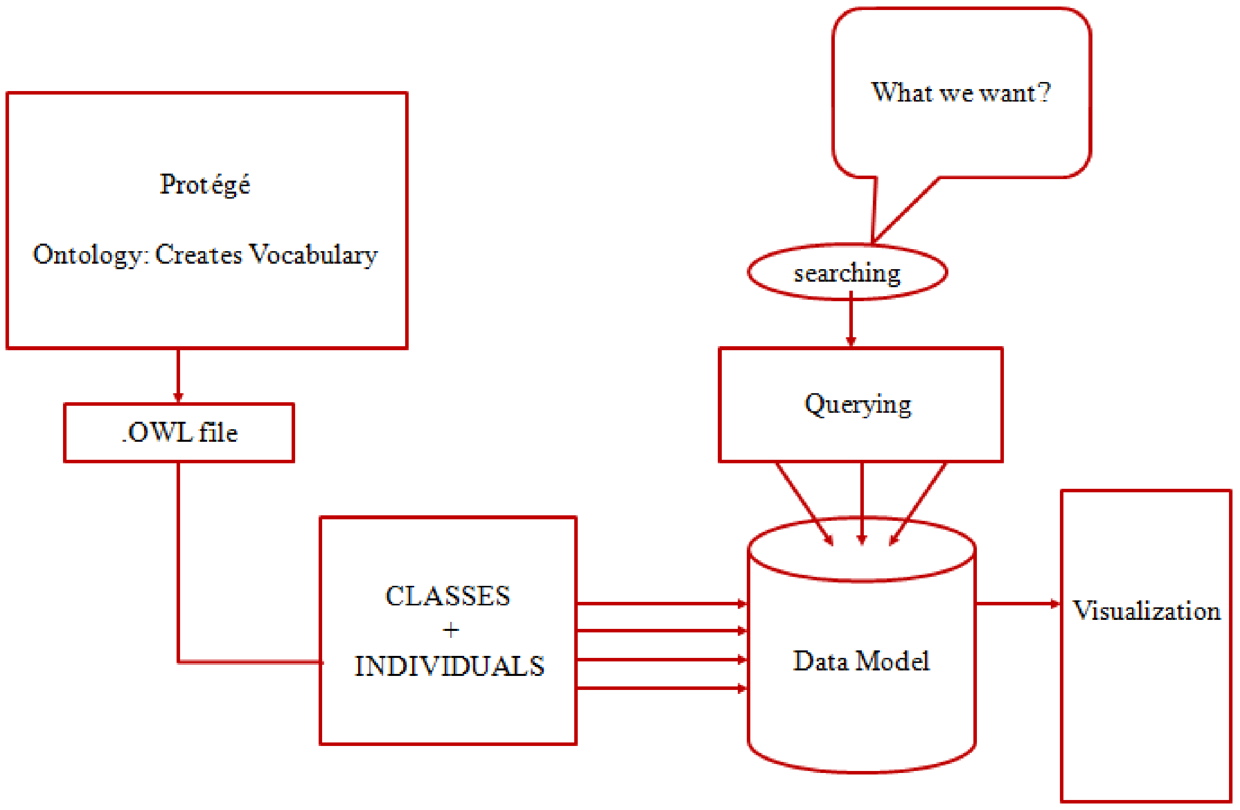


Fig-5.2.1 Block diagram of Computing Ontology

Terms are gathered, enumerated, categorized and defined with ontological properties among them for create an owl file using Protégé (section-4.5). This owl file containing classes and individuals form a data model.

The data model thus formed can be queried using ontology query languages like SPARQL.

5.2.1 Classes in Computing Ontology:

In the current ontology, computing domain is classified into Seventeen (17) top level areas which are collectively called L1-Areas which is now revised and an abstractive collection is terms having 7 top levels is proposed which is explained in section 5.4

1. Algorithms and Theory
2. Computer and Network Systems
3. Computer Education
4. Computer Graphics
5. Computer Hardware Organization
6. Discrete Structures
7. Ethical and Social Concepts
8. History of Computing
9. Information Topics
10. Intelligent Systems
11. Mathematical Collections
12. Programming Fundamentals
13. Programming Languages
14. Security
15. System and Project Management
16. Systems Development
17. User Interface

These terms are further classified into L-2 Sub-Areas, L3 , L4 and so on till L7 meaning the currently ontology is 7 levels deeper.

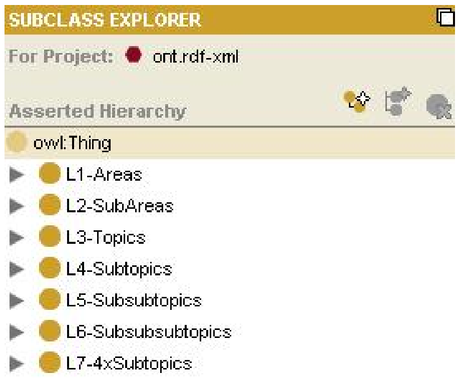
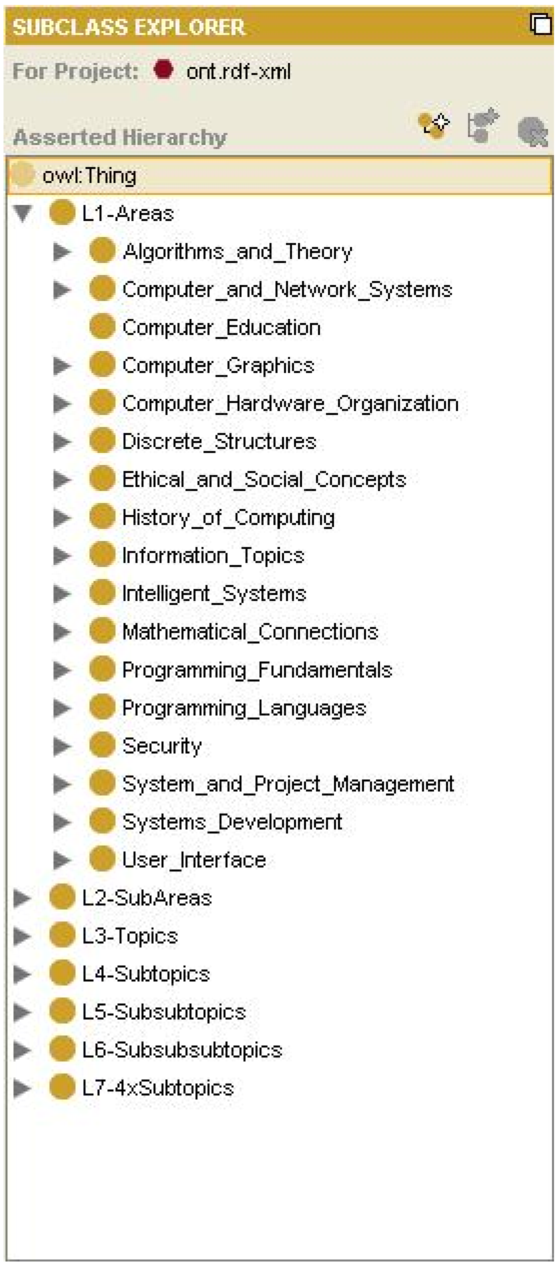
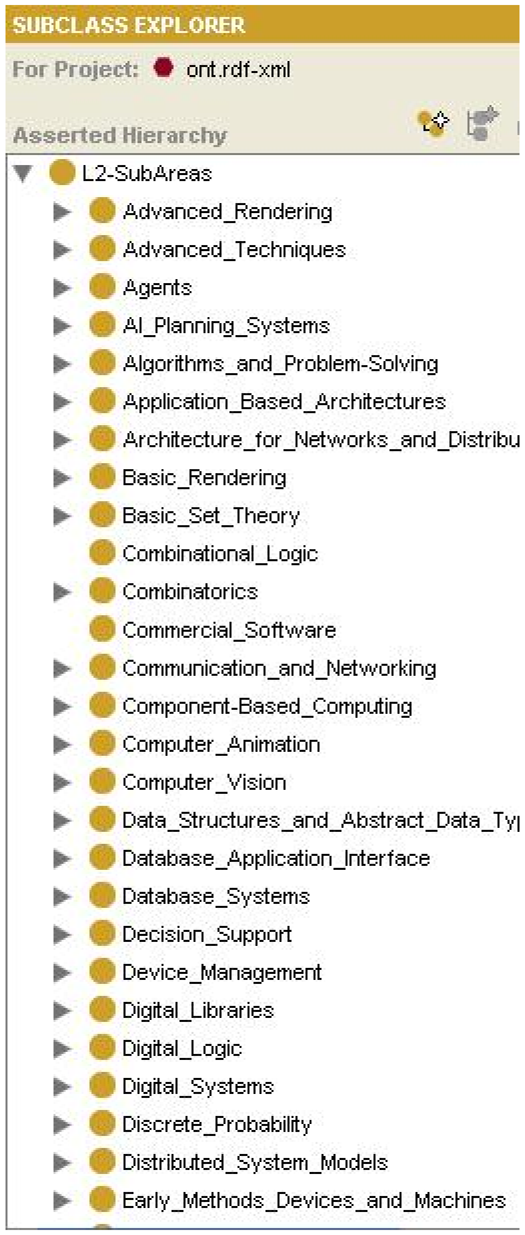


Table-5.2.2 Classification based on areas.

L1-Areas	L2-Sub Areas
	

SUBCLASS EXPLORER
For Project: ● ont.rdf-xml
Asserted Hierarchy

- owl:Thing
- ▼ L1-Areas
  - ▶ Algorithms_and_Theory
  - ▶ Computer_and_Network_Systems
  - Computer_Education
  - ▶ Computer_Graphics
  - ▶ Computer_Hardware_Organization
  - ▶ Discrete_Structures
  - ▶ Ethical_and_Social_Concepts
  - ▶ History_of_Computing
  - ▶ Information_Topics
  - ▶ Intelligent_Systems
  - ▶ Mathematical_Connections
  - ▶ Programming_Fundamentals
  - ▶ Programming_Languages
  - ▶ Security
  - ▶ System_and_Project_Management
  - ▶ Systems_Development
  - ▶ User_Interface
- ▶ L2-SubAreas
- ▶ L3-Topics
- ▶ L4-Subtopics
- ▶ L5-Subsubtopics
- ▶ L6-Subsubsubtopics
- ▶ L7-4xSubtopics

SUBCLASS EXPLORER
For Project: ● ont.rdf-xml
Asserted Hierarchy

- ▼ L2-SubAreas
  - ▶ Advanced_Rendering
  - ▶ Advanced_Techniques
  - ▶ Agents
  - ▶ AI_Planning_Systems
  - ▶ Algorithms_and_Problem-Solving
  - ▶ Application_Based_Architectures
  - ▶ Architecture_for_Networks_and_Distribu
  - ▶ Basic_Rendering
  - ▶ Basic_Set_Theory
  - Combinational_Logic
  - ▶ Combinatorics
  - Commercial_Software
  - ▶ Communication_and_Networking
  - ▶ Component-Based_Computing
  - ▶ Computer_Animation
  - ▶ Computer_Vision
  - ▶ Data_Structures_and_Abstract_Data_Ty
  - ▶ Database_Application_Interface
  - ▶ Database_Systems
  - ▶ Decision_Support
  - ▶ Device_Management
  - ▶ Digital_Libraries
  - ▶ Digital_Logic
  - ▶ Digital_Systems
  - ▶ Discrete_Probability
  - ▶ Distributed_System_Models
  - ▶ Early_Methods_Devices_and_Machines

Table-5.2.3 L1-Areas and L2-Sub Areas

The above table shows arrangement of terms into L1 and L2 similarly other levels are populated accordingly. There are more than 4000 terms.

5.2.2 Individuals in Computing Ontology

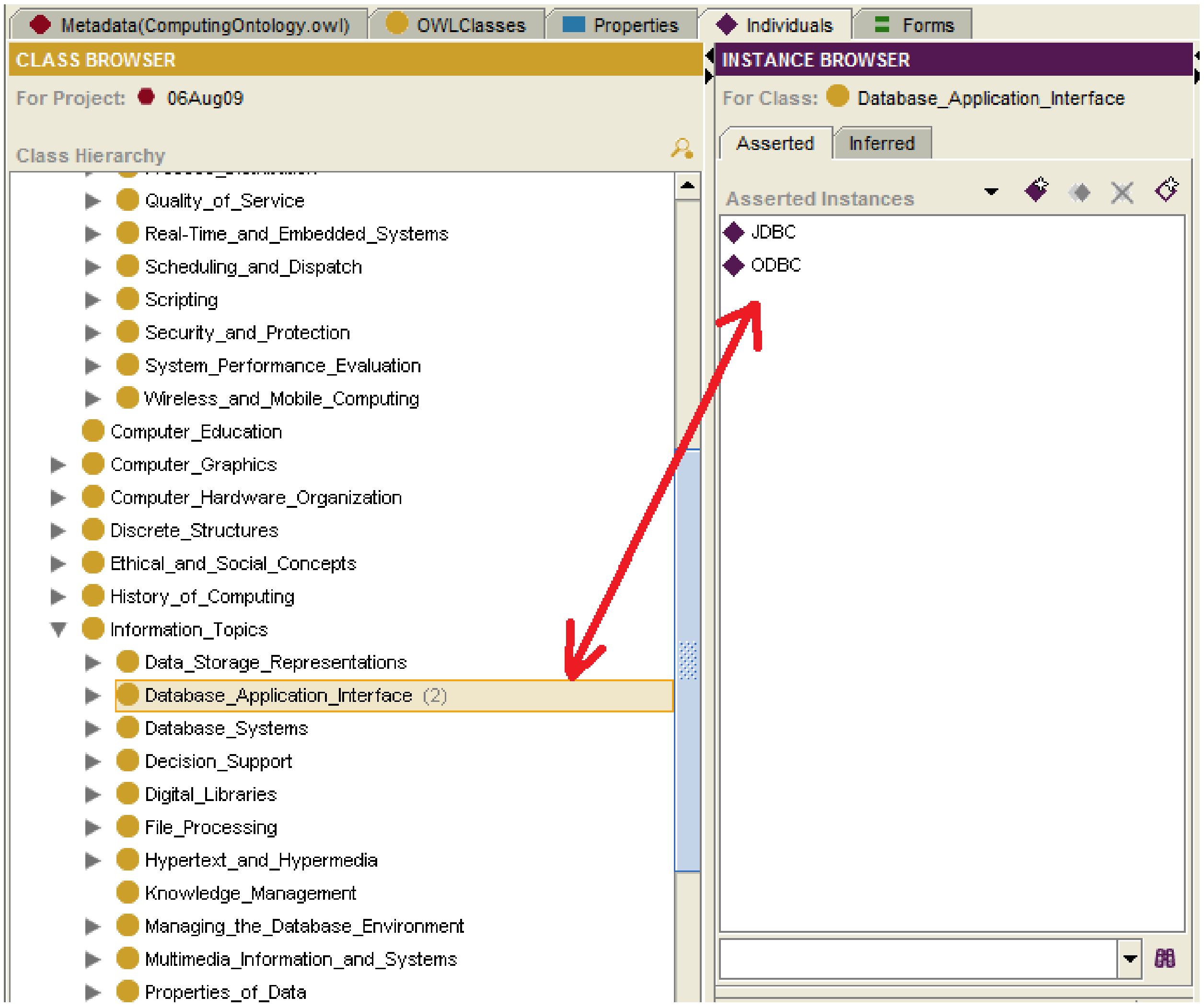


Table-5.2.4 Individuals in Computing Ontology

Table- 5.2.4 shows individuals associated to Database_Application_Interface. Individuals ate the instance of that that particular class or concept.

5.2.3 Structures of Computing Ontology

The Computing Ontology OWL file having these terms has the structure as:

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
    xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.owl-ontologies.com/ComputingOntology.owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:swrl="http://www.w3.org/2003/11/swrl#"
    xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.owl-ontologies.com/ComputingOntology.owl">
```

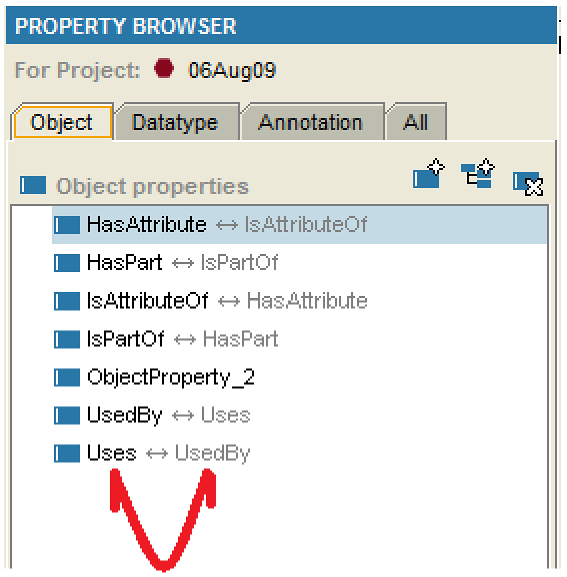
```
<owl:Ontology rdf:about=""/>
```

```
<owl:Class rdf:ID="project_staffing">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Project_personnel_and_organization"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="L4-Subtopics"/>
    </rdfs:subClassOf>
  </owl:Class>
```

```
<owl:Class rdf:ID="Telnet">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Protocol_Examples_for_Layer_5"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="L5-Subsubtopics"/>
    </rdfs:subClassOf>
  </owl:Class>
```

```
<owl:Class rdf:ID="centralized">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Internal_organizational_structures"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#L4-Subtopics"/>
  </owl:Class>
```

5.2.4 Properties in Computing Ontology



HasAttribute ↔ IsAttributeOf
HasPart ↔ IsPartOf
IsAttributeOf ↔ HasAttribute
IsPartOf ↔ HasPart
ObjectProperty_2
UsedBy ↔ Uses
Uses ↔ UsedBy

Table-5.2.5 Properties in Computing Ontology

Computing ontology has seven (7) Object type properties which also have corresponding inverse properties. For instance, a UsedBy property has Uses as its inverse property which brings both the classes together and access one for the other.

OWL Properties and Individuals defined as:

```
<owl:ObjectProperty rdf:about="#HasAttribute">
    <rdfs:range>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Definitions_of_Sets"/>
          <owl:Class rdf:about="#Definitions"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:range>
    <owl:inverseOf>
      <owl:ObjectProperty rdf:about="#IsAttributeOf"/>
    </owl:inverseOf>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="ObjectProperty_2"/>
  <owl:ObjectProperty rdf:about="#IsAttributeOf">
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Definitions"/>
          <owl:Class rdf:about="#Definitions_of_Sets"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <owl:inverseOf rdf:resource="#HasAttribute"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="DatatypeProperty_4"/>
  <owl:TransitiveProperty rdf:about="#UsedBy">
    <rdfs:range rdf:resource="#Hashing"/>
    <rdfs:domain rdf:resource="#Hashed_File_Organization"/>
    <owl:inverseOf>
      <owl:TransitiveProperty rdf:about="#Uses"/>
    </owl:inverseOf>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </owl:TransitiveProperty>
  <owl:TransitiveProperty rdf:about="#Uses">
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <owl:inverseOf rdf:resource="#UsedBy"/>
  </owl:TransitiveProperty>
  <owl:TransitiveProperty rdf:about="#IsPartOf">
```

```
    <owl:inverseOf>
      <owl:TransitiveProperty rdf:about="#HasPart"/>
    </owl:inverseOf>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </owl:TransitiveProperty>
  <owl:TransitiveProperty rdf:about="#HasPart">
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <owl:inverseOf rdf:resource="#IsPartOf"/>
  </owl:TransitiveProperty>
  <Compleixty_Classes_Based_on_Mode_of_Computing rdf:ID="Randomized"/>
  <Operators_on_Sets rdf:ID="Intersection"/>
  <Laws_Regarding_IT_in_Organizations rdf:ID="Sarbanes-Oxley"/>
  <Patterns_in_Data_Mining rdf:ID="Clustering_of_Patterns"/>
  <Reductions rdf:ID="Poly-time_Reductions"/>
  <Compleixty_Classes_Based_on_Mode_of_Computing
rdf:ID="Alternating"/>
  <Hardware_Description_Languages rdf:ID="VHDL"/>
  <Non_Comparison_Based_Sorts rdf:ID="Radix_Sort"/>
  <Operators_on_Sets rdf:ID="Union"/>
  <Normal_Form rdf:ID="BCNF"/>
  <Quadratic_Sorts rdf:ID="Selection_Sort"/>
  <nlogn_Sort rdf:ID="Merge_Sort"/>
  <Asymptotic_Analysis rdf:ID="Worst_Case_Complexity_Bound"/>
  <Markup_Languages rdf:ID="SGML"/>
  <Asymptotic_Analysis rdf:ID="Average_Case_Complexity_Bound"/>
  <Parsing rdf:ID="Yacc"/>
  <Operators_on_Sets rdf:ID="Cartesian_Products"/>
  <Specific_Modeling_Languages rdf:ID="ER_Diagram"/>
  <Types_of_Databases rdf:ID="Object-Oriented_Databases"/>
  <Specific_Modeling_Languages rdf:ID="UML_Class_Diagram"/>
  <Collision_Resolution_Techniques rdf:ID="Chaining"/>
  <Compleixty_Classes_Based_on_Mode_of_Computing rdf:ID="Online"/>
  <Normal_Form rdf:ID="Third_NF"/>
  <Special_Purpose_Databases rdf:ID="Multimedia_Databases"/>
  <Asymmetric_Cryptography rdf:ID="Public_Key_Encryption"/>
  <Geometric_Algorithms                         rdf:ID="Line_Segments_-
_Properties_or_Intersections"/>
  <Search rdf:ID="Binary_Search"/>
  <Operators_on_Sets rdf:ID="Power_Sets"/>
  <Geometric_Algorithms rdf:ID="Other"/>
  <Normal_Form rdf:ID="First_NF"/>
  <Search rdf:ID="Hash-based_Retrieval"/>
  <Transitive_Closure rdf:ID="Dijkstras_Algorithms"/>
  <Famous_NP_Complete_Problems rdf:ID="Vertex_Cover"/>
  <Transitive_Closure rdf:ID="Floyds_Algorithm"/>
  <Collision_Resolution_Techniques rdf:ID="Clustering"/>
  <Search rdf:ID="Interpolation_Search"/>
  <Markup_Languages rdf:ID="XML"/>
  <Reductions rdf:ID="Log-space_Reductions"/>
  <Non_Comparison_Based_Sorts rdf:ID="Bucket_Sort"/>
  <Special_Purpose_Databases rdf:ID="Statistical_Databases"/>
  <Search rdf:ID="Jump_Search"/>
  <Special_Purpose_Databases rdf:ID="Scientific_Databases"/>
  <State-based_and_Table-driven_Construction rdf:ID="Lex"/>
  <Famous_NP_Complete_Problems rdf:ID="Traveling_Salesperson"/>
  <Turing_Machines rdf:ID="Universal_Turing_Machine"/>
  <Transforms rdf:ID="FFT"/>
  <Patterns_in_Data_Mining rdf:ID="Frequent_Sets"/>
  <Markup_Languages rdf:ID="HTML"/>
  <Empirical_Measurements_of_Performance rdf:ID="Profiling_Programs"/>
  <Normal_Form rdf:ID="Fourth_NF"/>
  <Types_of_Databases rdf:ID="Rule-Based_Databases"/>
  <Normal_Form rdf:ID="Second_NF"/>
  <nlogn_Sort rdf:ID="Quick_Sort"/>
  <Special_Purpose_Databases rdf:ID="Textual_Databases"/>
  <Famous_NP_Complete_Problems rdf:ID="Integer_Programming"/>
  <Search rdf:ID="Sequential_Search"/>
  <Quadratic_Sorts rdf:ID="Insertion_Sort"/>
  <Types_of_Databases rdf:ID="Relational_Databases"/>
  <Turing_Machines rdf:ID="Nondeterministic_Turing_Machines"/>
  <Database_Content rdf:ID="Tuples"/>
  <Special_Purpose_Databases rdf:ID="Spatial_Databases_and_GIS"/>
  <Collision_Resolution_Techniques rdf:ID="Probing"/>
  <Normal_Form rdf:ID="Fifth_NF"/>
  <Compleixty_Classes_Based_on_Mode_of_Computing
rdf:ID="Interactive"/>
  <Types_of_Databases rdf:ID="XML_Databases"/>
  <Unkeyed_Cryptography rdf:ID="Parabolic_Encryption"/>
  <Special_Purpose_Databases rdf:ID="Image_Databases"/>
  <Compleixty_Classes_Based_on_Mode_of_Computing rdf:ID="Parallel"/>
  <Symmetric_Cryptography rdf:ID="Secret_Key_Encryption"/>
  <nlogn_Sort rdf:ID="Heap_Sort"/>
  <Famous_NP_Complete_Problems rdf:ID="Three_SAT"/>
  <Object-oriented_Run-time_Issues rdf:ID="Dynamic_Binding"/>
  <Special_Purpose_Databases rdf:ID="Temporal_Databases"/>
  <Patterns_in_Data_Mining rdf:ID="Association_Rules"/>
  <Sorting rdf:ID="Quadratic_Sorting"/>
  <Operators_on_Sets rdf:ID="Complement"/>
  <Geometric_Algorithms rdf:ID="Convex_Hull_Finding_Algorithms"/>
  <Logics_of_Uncertainty rdf:ID="Fuzzy_Logic"/>
  <Database_Application_Interface rdf:ID="JDBC"/>
  <Database_Application_Interface rdf:ID="ODBC"/>
</rdf:RDF>
```

5.3 Ontology Visualization

The purpose of Ontology visualization is to provide a GUI to navigate, observe and understand the bridges between various areas of computing ontology.

A third party tool called The Brain::Personal Brain[27] is used to visualize the ontology. Personal Brian is one of the most effective tools for mapping hierarchal and complexly related structures. The owl file is parsed thorough personal brain and the following visualization is achieved.

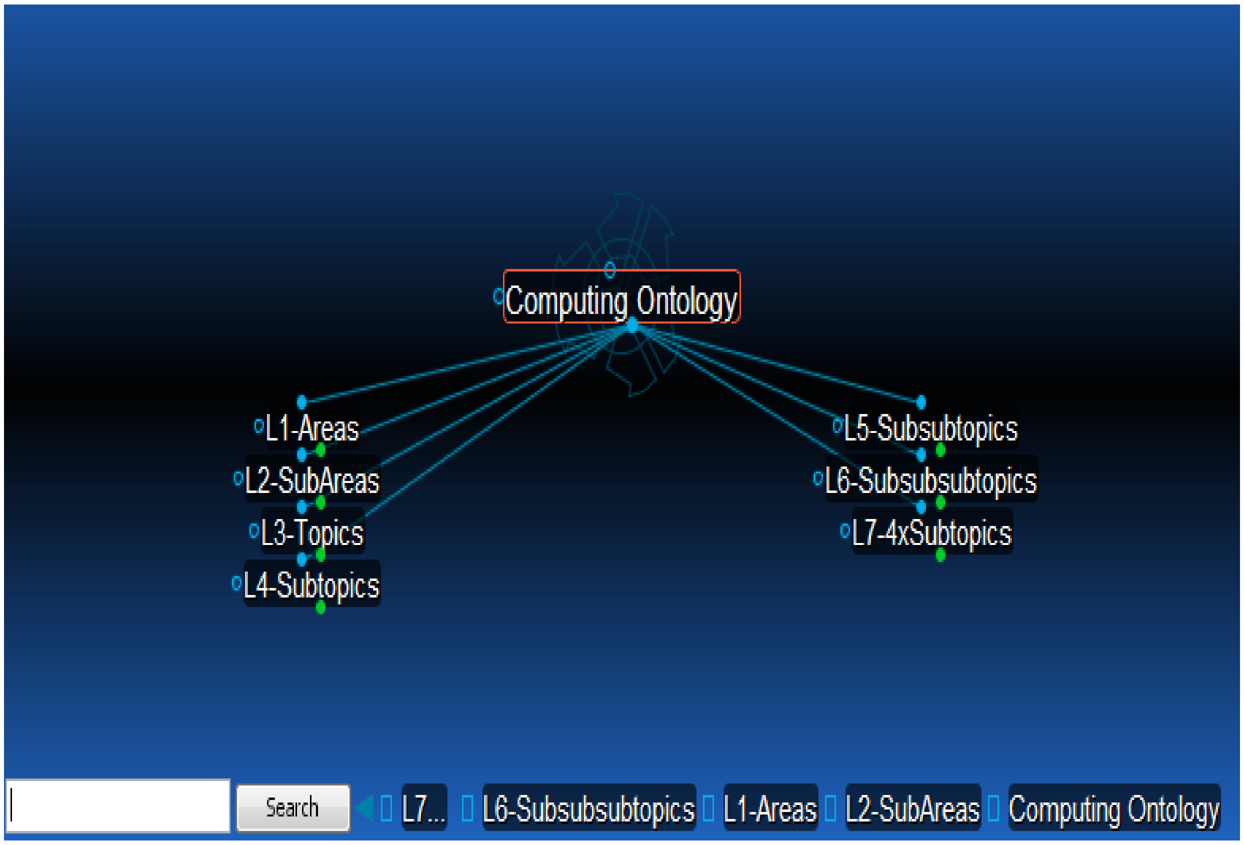


Fig-5.3.1-Visualization of Base Ontology

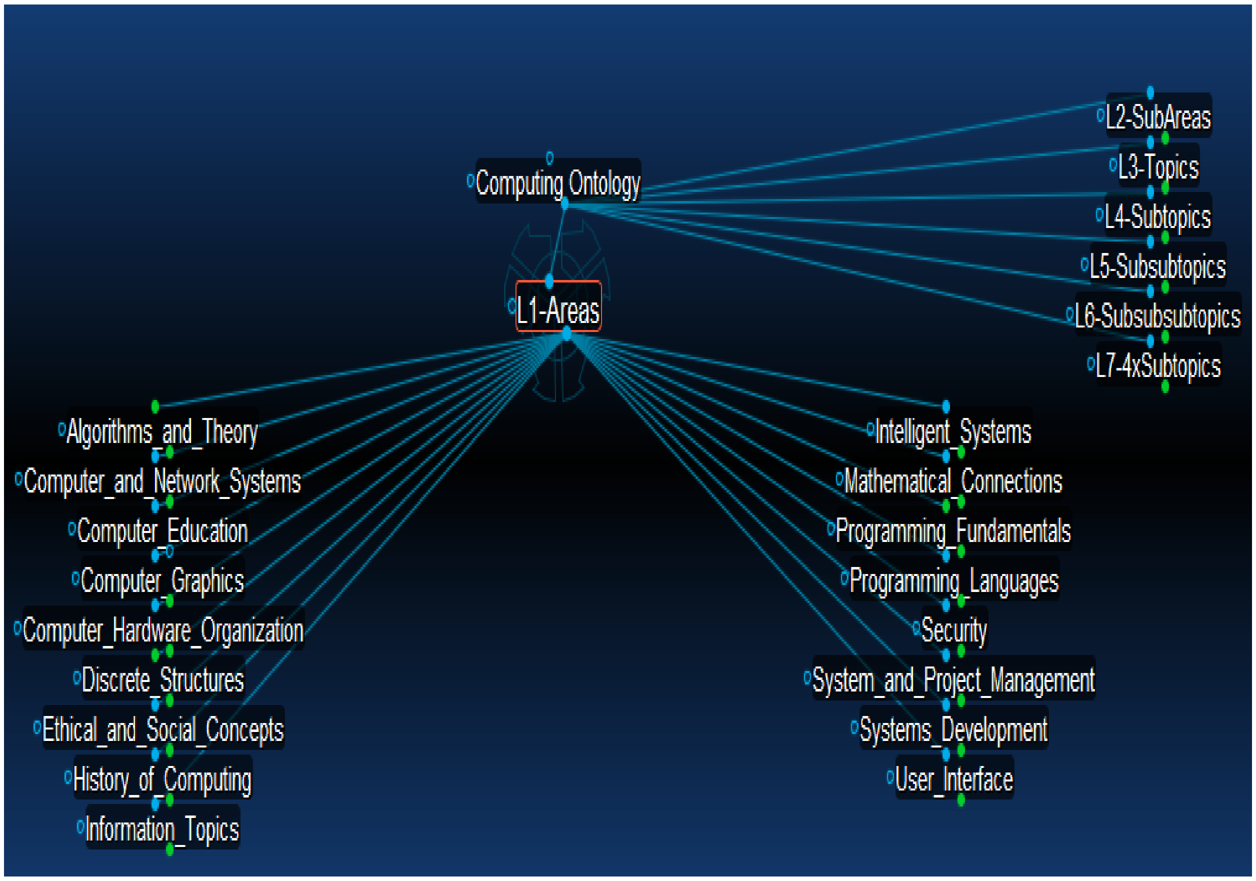


Fig-5.3.2 L1- Areas: the top 17 Classes.

Fig- 5.3.2 shows the top 17 classes while at the same time showing the sibling classes to L1-Areas. The structure allows navigating deep while exploring the jump connections as shown in Fig-5.3.3.

Fig- 5.3.3, shows Algorithms to be at the root of visualization and it is shows the associated parent, child and siblings.

Algorithm's is present in L3-Topics of the ontology. Algorithm's is also child of Recursive Expressions implies it has two direct parents.
Algorithms in turn has seven (7) children which are present be in L4-Subtopics, for instance Advanced_Data_Structures is child for Algorithm's present in L4-Subtopics as shown in Fig-5.3.4.

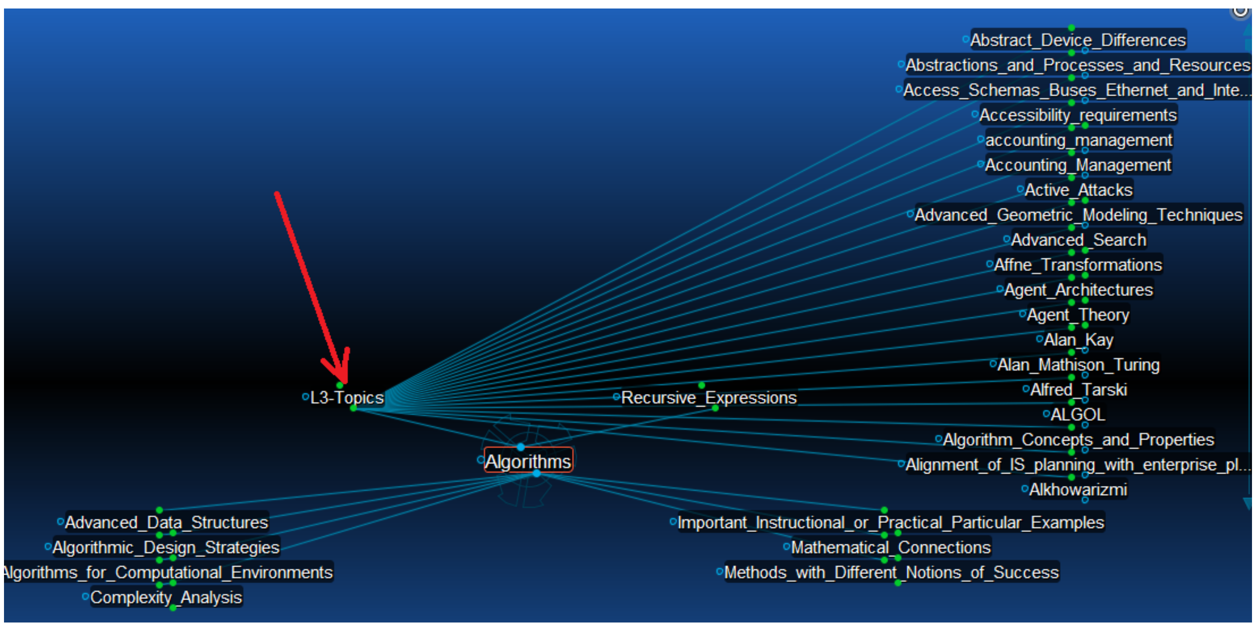

Abstract_Device_Differences
Abstractions_and_Processes_and_Resources
Access_Schemas_Buses_Ethernet_and_Inte...
Accessibility_requirements
accounting_management
Accounting_Management
Active_Attacks
Advanced_Geometric_Modeling_Techniques
Advanced_Search
Affne_Transformations
Agent_Architectures
Agent_Theory
Alan_Kay
Alan_Mathison_Turing
L3-Topics
Recursive_Expressions
Alfred_Tarski
ALGOL
Algorithm_Concepts_and_Properties
Algorithms
Alignment_of_IS_planning_with_enterprise_pl...
Alkhowarizmi
Advanced_Data_Structures
Algorithmic_Design_Strategies
lgorithms_for_Computational_Environments
Complexity_Analysis
Important_Instructional_or_Practical_Particular_Examples
Mathematical_Connections
Methods_with_Different_Notions_of_Success

Fig-5.3.3 Algorithms- L3-Topics

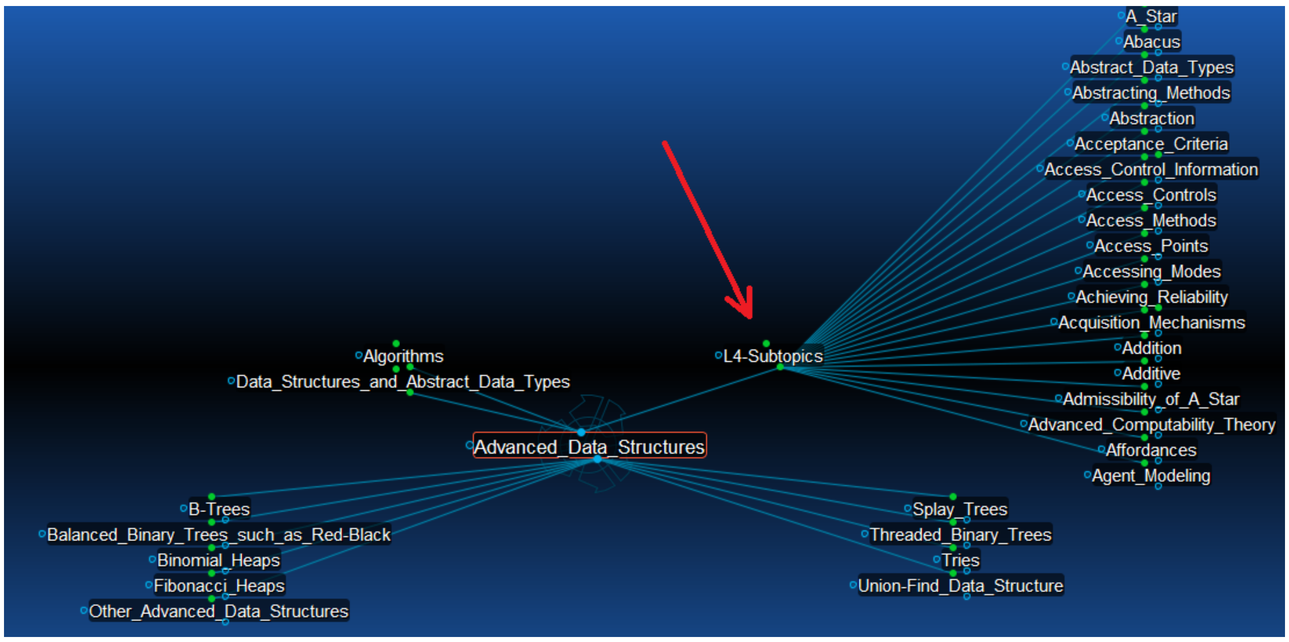


Fig-5.3.4 Algorithms- L4-Subtopics

5.4 Future Scope:

The current ontology as described above have 17 top level classes with 7 levels deep. This is now in process of abstraction. OWL file is created and is available as an initial release to play with.
Work is being done to create more abstracted ontology where the computing domain is categorized into 7 top classes instead of 17.

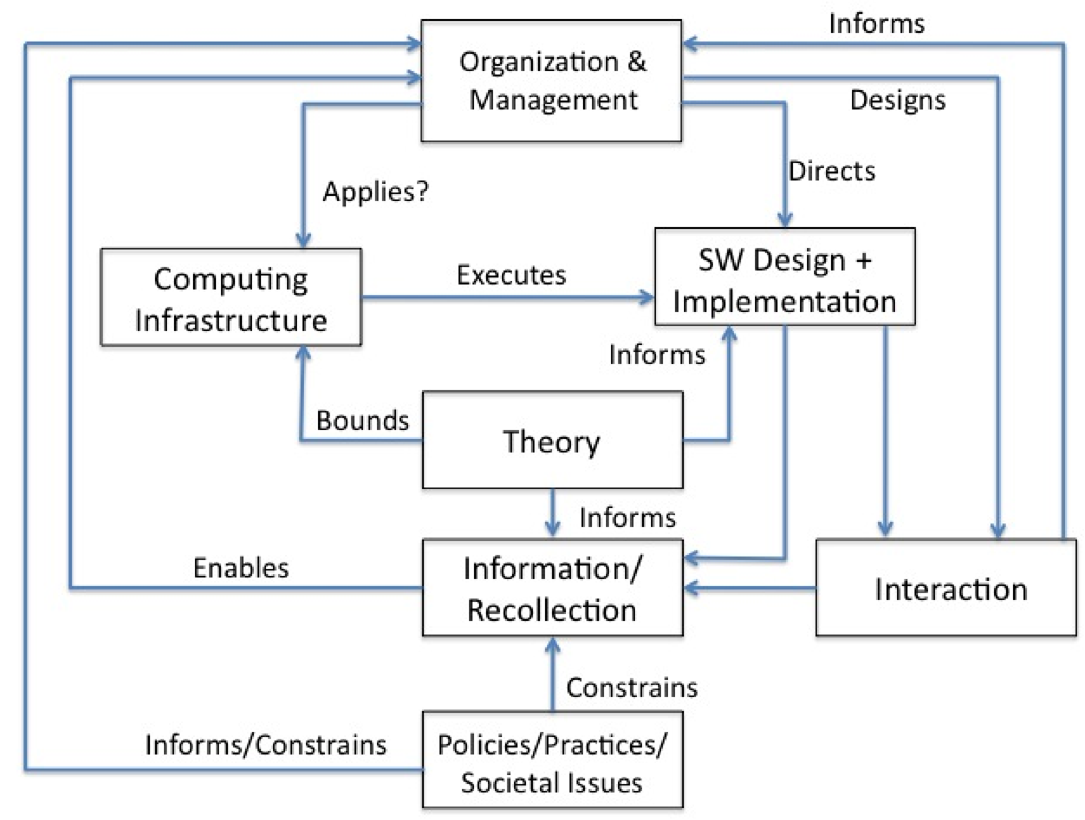


Fig-5.4.1 New classification

These 7 classes incorporate all the contents of the old classification. Domain experts have tabulated terms.

6. Conclusion

In this paper we have addressed different aspects of Semantic Web and ontologies which are the primary building blocks of Semantic Web. We also proposed ODLC (Ontology Development Life Cycle) which a software engineering perspective of designing ontologies. Finally, we introduced Computing Ontology, an ontology of all computing domain designed in ODLC.
More about Computing Ontology can be found at the location
http://www.distributedexpertise.org/computingontology/

References:

[1] **Tim Berners-Lee**, **Robert Cailliau**, **Jean-François Groff**, **Bernd Pollermann**, (1992) "World-Wide Web: The Information Universe", Internet Research, Vol. 2 Iss: 1, pp.52 – 58

[2] The History Of World Wide Web. Wikipedia. http://en.wikipedia.org/wiki/History_of_the_World_Wide_Web.

[3] **T.Berners-Lee**, **J Hendler**, **O. Lassila**, Scientific American: The Semantic Web, May 27,2001 May, pp. 28-37. http://www.scientificamerican.com/article.cfm?id=the-semantic-web

[4] Purpose of using ontologies in software engineering http://people.cis.ksu.edu/~abreed/CIS890/Sft_Engineering.pdf

[5] **Pornpit Wongthongtham**, **Elizabeth Chang**, **Ian Sommerville**, Digital Ecosystems and Business Intelligence Institute, Curtin University, Australia. http://protege.stanford.edu/conference/2007/presentations/04.02_Wongthongtham.pdf

[6] **Olavo Mendes**, **Alain Abra**, Software Engineering Ontology: A Development Methodology. http://logic.csci.unt.edu/tarau/teaching/SoftEng/OLD/SEOntologies/839.pdf

[7] Ontology as software Artifacts http://people.cis.ksu.edu/~abreed/CIS890/Software_Artifacts.pdf

[8] **Michela Chimienti**, **Michele Dassisti**, **M. Missikoff**, **A. De Nicola**, Benchmarking Criteria to evaluate Ontology building methodologies, Open Interop Workshop on Enterprise Modeling and Ontology for Interoperability EMOI – INTEROP'06, June(2006) http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-200/14.pdf

[9] **Antonio De Nicola**, **Michele Missikoff**, **Roberto Navigli**, A Software Engineering approach to ontology building, Information Systems 34 (2009) pp 258-275 http://www.dsi.uniroma1.it/~navigli/pubs/De_Nicola_Missikoff_Navigli_2009.pdf

[10] **Hans-Jorg Happel**, **Stefan Seedorf**, Applciations of ontologies in Software Engineering, International workshop on Semantic Web Enabled Software Engineering SWESE'06 Athens, USA; Nov(2006)

[11] **Sinuhé Arroyo**, **Rubén Lara**, **Ying Ding**, **Michael Stollberg**, **Dieter Fensel** SEMANTIC WEB LANGUAGES – STRENGTHS AND WEAKNESS, European Commission in the context of the Esperonto Services project, IST-2001-34373

[12] **Mills Davis**, Project 10x Semantic Wave 2008 Report: Industry Roadmap to Web 3.0 & Multibillion Dollar Market Opportunities, (October 2008) http://www.isoco.com/pdf/Semantic_Wave_2008-Executive_summary.pdf

[13] **Peter F. Patel-Schneider**, **J´erome Sim´eon**, Building the Semantic Web on XML, ISWC 2002, LNCS 2342, pp. 147–161, 2002
http://www.mpi-inf.mpg.de/departments/d5//teaching/ss03/xml-seminar/Material/PS02.pdf

[14] **Haytham Al-Feel**, **M.A.Koutb**, **Hoda Suoror**, Toward An Agreement on Semantic Web Architecture, World Academy of Science, Engineering and Technology 49 (2009)
http://www.waset.org/journals/waset/v49/v49-142.pdf

[15] **Sinuhé Arroyo**, **Rubén Lara**, **Ying Ding**, **Michael Stollberg**, **Dieter Fensel**, SEMANTIC WEB LANGUAGES – STRENGTHS AND WEAKNESS.

[16] **Lee W. Lacy** OWL- Representing Information Using the Web Ontology Language, January, 2005 Ch-5, URI's and Namespaces, pp-49.

[17] **Lee W. Lacy** OWL- Representing Information Using the Web Ontology Language, January, 2005 Ch-6, XML, pp-63.

[18] **Lee W. Lacy** OWL- Representing Information Using the Web Ontology Language, January, 2005 Ch-6, XML, pp-73.

[19] **Asuncion Gomez-perez**, **Mariano Fernandez-Lopez**, **Oscar Corcho**, Ontological Engineering, Springer, (2004), Ch-1, Throretical Foundation of Ontologies, pp-7

[20] Ontology Development 101: A Guide to Creating Your First Ontology http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html

[21] **Lee W. Lacy** OWL- Representing Information Using the Web Ontology Language, January, 2005 Ch-3, Sec-3.1.3, pp-27

[22] **Lee W. Lacy** OWL- Representing Information Using the Web Ontology Language, January, 2005 Ch-3, Sec-3.1.1, pp-26

[23] **Annika Ohgren**, **Kurt Sandkuhl**, Towards a Methodology for Ontology Development in Small and Medium-sized Enterprises, IADIS Interntional Conference on Applied Computing 2005, pp-369-376
http://www.iadis.net/dl/final_uploads/200501L044.pdf

[24] OWL Web Ontology Language Guide, (Feb 2004) http://www.w3.org/TR/owl-guide/

[25] OWL Web Ontology Language Reference, (Feb 2004) http://www.w3.org/TR/owl-ref/

[26] Protégé: open source ontology editor http://protege.stanford.edu/doc/users.html

[27] Personal Brain-6 user documentation
http://assets.thebrain.com/documents/PersonalBrain-6-Guide.pdf