Semantic Search

M.Tech Seminar Report

Submitted by

Prashanth Kamle
Roll No: 08305006

Under the guidance of

Prof. Pushpak Bhattacharyya

Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Mumbai

2009

Abstract

Keyword based search retrieves documents based on presence of keywords in the document. This, even though effective, does not give relevant results when the meaning of the query conveys more information about what the user wants than the actual choice of words. Semantic search techniques attempt to solve this problem by annotating documents with concepts from ontologies, thus creating a knowledge base. Retrieval of documents or portions of documents based on the query is carried out by performing inferencing on the knowledge base.

Acknowledgments

I would like to thank my guide Prof. Pushpak Bhattacharyya for his invaluable support and guidance.

Table of Contents

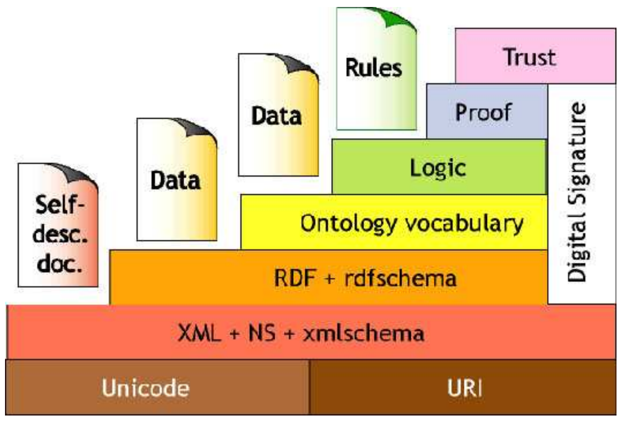


Figure 1: Semantic web architecture

1 Introduction

1.1 Overview

In the early 1990s, Tim Berners-Lee[3] spent a frustrating year trying to make people understand the power and beauty of his idea of an internet hypertext system, which he named the World Wide Web. A decade later, he is struggling with the same problem with his idea of the Semantic Web. The word *semantic* implies meaning, or as Wordnet defines it "of or relating to meaning or the study of meaning". The idea is a web that not only interconnects documents with one another, but also recognizes the meaning of the information in those documents.

The ability to perform a semantic search makes the semantic web very powerful. As an example, consider the query "is Taj Mahal taller than Eiffel tower?". A semantic search engine understands that the term "tall" relates to height, fetches the heights of Taj Mahal and Eiffel Tower from semantically annotated pages on the internet, performs a comparison and returns "no" as result.

Adding semantics will radically change the nature of the web - from a place where information is merely displayed to one where it is interpreted, exchanged and processed. Semantic search engines will be able to collect machine-readable data from diverse sources, process it and infer new facts.

1.2 Semantic Search

To make semantic search possible, the following things need to be done.

- Develop a super-language which is powerful enough to express all human knowledge, and is machine understandable as well.
- Convince all website owners to represent their information in this language.
- Write programs that can search and reason about the information on the web.

Tim Berners-Lee proposed the architecture shown in Figure 1 for the semantic web. He outlined the architecture of the Semantic Web in the following three layers:

1. The metadata layer. The data model at this layer contains just the concepts of resource and properties. Currently, the RDF (Resource Description Framework) is believed to be the most popular data model for the metadata layer.
2. The schema layer. Web ontology languages are introduced at this layer to define a hierarchical description of concepts (is-a hierarchy) and properties. Currently, RDFS (RDF Schema) is considered as a candidate schema layer language.

3. The logical layer. More powerful web ontology languages are introduced at this layer. These languages provide a richer set of modeling primitives that can be mapped to the well-known expressive Description Logics. Currently, OWL(Web Ontology Language) is the W3C recommendation for the logical layer.

This report describes in detail Ontologies - which form the super-language mentioned earlier, Description Logic - the language used to express ontologies, OWL - web ontology language and Querying.

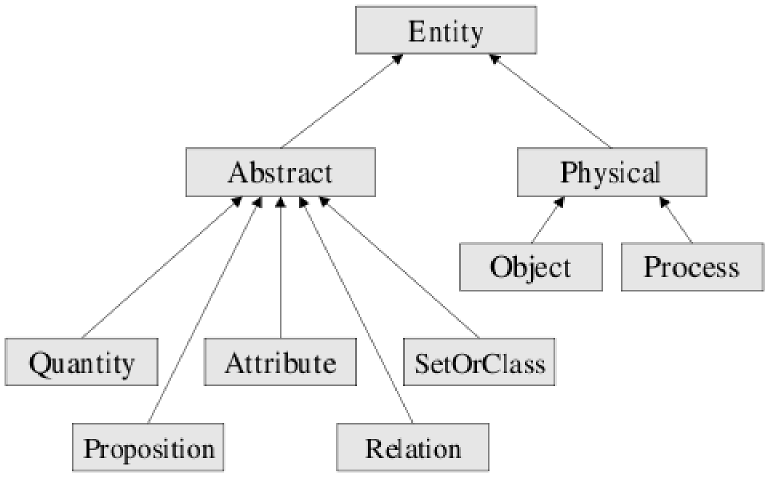


Figure 2: Top level concepts in an Ontology

2 Ontologies

2.1 Introduction

As Wikipedia[7] defines it, an ontology is a formal representation of a set of concepts within a domain and the relationships between those concepts. It is used to reason about the properties of that domain, and may be used to define the domain. Ontologies are used in artificial intelligence, the Semantic Web, software engineering, biomedical informatics, library science, and information architecture as a form of knowledge representation about the world or some part of it.

Components of an ontology may include

- Individuals: instances or objects
- Classes: concepts or types of objects
- Attributes: properties that individuals and classes can have
- Relations: ways in which individuals or classes can be related to one another
- Axioms: rules in a logical form that together form the theory that the ontology describes in its domain of application

2.2 Domain ontologies and upper ontologies

A domain ontology models a specific domain. For example, there can be ontologies of Communications, Countries and Regions, Distributed computing, Economy, Finance, Engineering components, Geography, Government, Military etc.

An upper ontology is a domain-independent ontology, intended to be reused and extended for a particular domain to form a domain ontology. It is a model of the common objects that are generally applicable across a wide range of domain ontologies. There are several standardized upper ontologies available for use, including Dublin Core, OpenCyc and SUMO.

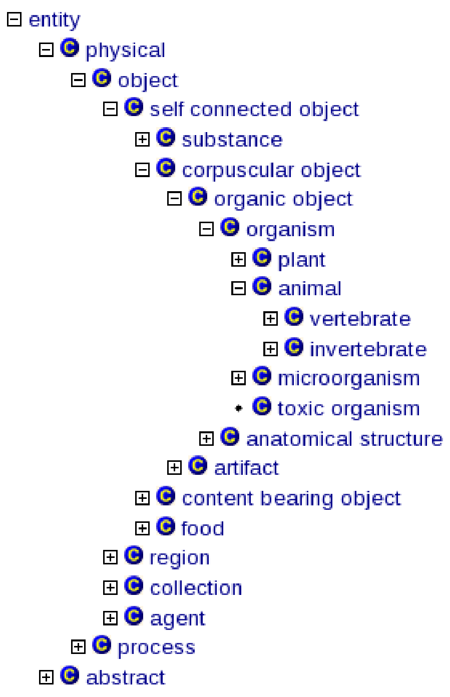


Figure 3: A concept hierarchy

2.3 Ontology Languages

An Ontology is expressed in an Ontology language. Some ontology languages are

- CycL
- KIF (Knowledge Interchange Format)
- FLogic (frame based)
- RDF (Resource Description Framework) and RDFS (RDF Schema)
- OIL (Ontology Inference Layer)
- OWL (Web Ontology Language)

The next section looks at SUMO[5] as an example for Ontology.

2.4 SUMO - Suggested Upper Merged Ontology

SUMO is an Upper Ontology, and hence can be used as a domain-independent substrate for designing domain ontologies. SUMO and its domain ontologies form the largest formal public ontology in existence today. SUMO was created by merging publicly available ontological content into a single, comprehensive, and cohesive structure.

The top level concepts in SUMO are shown in Figure 2

The concept hierarchy of SUMO for concept *animal* is in Figure 3

SUMO is written entirely in First Order Logic, using KIF. In KIF, an axiom is written as under -

```
(=>
       (instance ?DRIVE Driving)
       (exists (?VEHICLE)
                (and
                        (instance ?VEHICLE Vehicle)
                        (patient ?DRIVE ?VEHICLE))))
```

This axiom represents the fact "If there is an instance of Driving, there is a Vehicle that participates in that action". SUMO contains many such axioms and concepts. These axioms are used for inferencing.

Since First Order Logic is undecidable, it poses problems in inferencing from the Ontology. Hence, Description Logic, which is a subset of First Order Logic with decidable inference, is used to write Ontologies.

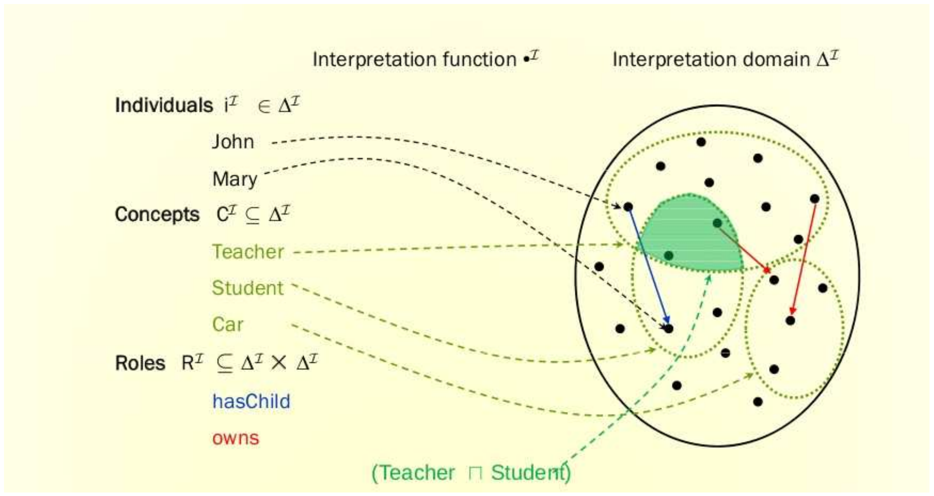

Interpretation function $\bullet^{\mathcal{I}}$
Interpretation domain $\Delta^{\mathcal{I}}$
Individuals $i^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
John
Mary
Concepts $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Teacher
Student
Car
Roles $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
hasChild
owns
(Teacher $\sqcap$ Student)

Figure 4: Concepts, roles and interpretation. Image taken from [8]

3 Description Logics

3.1 Introduction

Description Logics(DL)[2] is a family of languages for formal knowledge representation. DL represents knowledge by defining relevant concepts and roles in the domain. These definitions are called Terminology. Using terminology, the world description is specified in terms of properties of individuals and objects. These properties are called Assertions. DL is a subset of First Order Logic with decidable inference, and has been designed keeping in mind reasoning as a central service.

3.2 Constituents

The building blocks of description logics are

Concepts Unary predicates *Eg. Person, Female*

Roles Binary predicates *Eg. hasChild*

Individuals Constants *Eg. Mary, John*

Constructors
- Union $\sqcup$: *Eg.* $Man \sqcup Woman$
- Intersection $\sqcap$: *Eg.* $Person \sqcap Female$
- Restriction Exists $\exists$: *Eg.* $\exists hasChild.Female$
- Restriction ForAll $\forall$: *Eg.* $\forall hasChild.Engineer$
- Negation $\neg$: *Eg.* $\neg Man$
- Number restriction: $\leq k, \geq m$

Axioms Mother $\sqsubseteq$ Parent

For example,

$$Human \sqcap \neg Female \sqcap \exists married.Doctor \sqcap (\geq 5 child) \sqcap \forall child.Professor$$

represents “A man that is married to a doctor and has at least five children, all of whom are professors”

3.3 The DL family

The smallest DL family[8] is $\mathcal{ALC}$. Concepts are construted using $\sqcup, \sqcap, \neg, \forall, \exists$, but roles are restricted to atomic (no union/intersection/negation of roles allowed). Other members of the family are extensions of $\mathcal{ALC}$, and are named by adding more letters to the string $\mathcal{ALC}$.

- $\mathcal{S}$ is an abbreviation for $\mathcal{ALC}$ with transitive roles.
- $\mathcal{H}$ for role hierarchy. Eg: $hasDaughter \sqsubseteq hasChild$
- $\mathcal{O}$ for nominals. Eg: $\{Mary, Hohn\}$
- $\mathcal{I}$ for inverse roles. Eg: $isChildOf \equiv hasChild^{-1}$
- $\mathcal{N}$ for cardinality restrictions. Eg: $\geq 2hasChild$
- $\mathcal{F}$ restricts cardinality to be 0 or 1.
- $\mathcal{Q}$ for qualified number restrictions. Eg: $\geq 2hasChild.Professor$
- $\mathcal{R}$ for role inclusion and role disjointness.
- $(\mathcal{D})$ to denote use of datatype properties, data values or data types.

For example, $\mathcal{SHOIQ} = \mathcal{ALC}$ with transitive roles $+\mathcal{H} + \mathcal{O} + \mathcal{I} + \mathcal{Q}$

3.4 DL Interpretation

- An interpretation $\mathcal{I}$ is a tuple$(\triangle^I, \cdot^I)$ where
  - $\triangle^I$ is the domain
  - $\cdot^I$ is a mapping which maps
    - Names of individuals to elements of $\triangle^I$
    - Names of concepts to subsets of $\triangle^I$
    - Names of roles to subsets of $\triangle^I \times \triangle^I$

Figure 4 illustrates concepts, roles and interpretation.

3.5 The knowledge base

The knowledge base consist of Terminologies - the TBox, and Assertions - the ABox. Figure 5 shows the architecture of a DL knowledge base. The TBox includes Inclusion axioms and Equivalence axioms. A sample TBox is as under.

$$\begin{aligned}
Woman &\equiv Person \sqcap Female\\
Man &\equiv Person \sqcap \neg Woman\\
Mother &\equiv Woman \sqcap \exists hasChild.Person\\
Father &\equiv Man \sqcap \exists hasChild.Person\\
Parent &\sqsubseteq Person
\end{aligned}$$

Axioms with $\equiv$ are equivalence axioms, while axioms with $\sqsubseteq$ are inclusion axioms.

A sample ABox is below.

$$\begin{aligned}
&Father(PETER)\\
&Mother(MARY)\\
&hasChild(MARY, PETER)\\
&hasChild(PETER, HARRY)
\end{aligned}$$

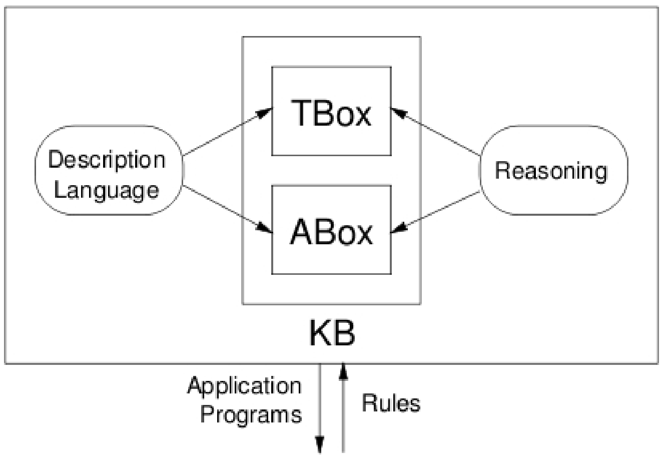


Figure 5: DL Knowledge base

3.6 Inferencing

The following kinds of inferences can be drawn from the knowledge base

1. Satisfiability: Is there some interpretation that satisfies axioms in TBox?
2. Subsumption: Is concept A more general than concept B?
3. Equivalence: Are concept A and concept B the same?
4. Instance check: Can assertion α be entailed by the ABox?
5. Retrieval: Which individuals satisfy concept C?

All the above inferences can be reduced to performing satisfiability test.

3.6.1 Tableaux Inferencing algorithm

This algorithm uses negation to reduce subsumption to a satisfiability problem, i.e $C \sqsubseteq D$ iff $C \sqcap \neg D$ is unsatisfiable. The central idea of the algorithm is to find an individual which satisfies the set of concepts. The algorithm is as under.

1. Convert description to Negation Normal form
2. For any existential restriction, introduce a new individual as role filler such that it satisfies the constraints expressed by the restriction.
3. Use value restrictions in interaction with already defined role relationships to impose new constraints on individuals
4. For disjunctive constraints, try both possibilities in successive attempts. Backtrack if you reach an obvious contradiction
5. If an at-most number restriction is violated then the algorithm must identify different role fillers

Let us run this algorithm to check whether $(\exists R.A) \sqcap (\exists R.B)$ is subsumed by $\exists R.(A \sqcap B)$.

1. The concept to check for satisfiability is

$$C = (\exists R.A) \sqcap (\exists R.B) \sqcap \neg(\exists R.(A \sqcap B))$$

   If it is unsatisfiable, then $(\exists R.A) \sqcap (\exists R.B) \sqsubseteq (\exists R.(A \sqcap B))$

2. Move the negations as far inside as possible.

$$C = (\exists R.A) \sqcap (\exists R.B) \sqcap \forall R.(\neg A \sqcup \neg B)$$

   C is now in negation normal form.

3. Now, we try to construct an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \phi$. This means there must exist an individual in $\Delta^{\mathcal{I}}$ that is an element of $C^{\mathcal{I}}$. So, we construct an individual $b \in C^{\mathcal{I}}$.

4. Since C is the conjunction of 3 concepts, b must satisfy $b \in (\exists R.A)^{\mathcal{I}}, b \in (\exists R.B)^{\mathcal{I}}$ and $b \in (\forall R.(\neg A \sqcup \neg B))^{\mathcal{I}}$.

5. From $b \in (\exists R.A)^{\mathcal{I}}$, we can see that there must exist an individual c such that $(b, c) \in R^{\mathcal{I}}$ and $c \in A^{\mathcal{I}}$. Similarly, $b \in (\exists R.B)^{\mathcal{I}}$ implies that there must exist an individual d with $(b, d) \in R^{\mathcal{I}}$ and $d \in B^{\mathcal{I}}$.

6. Since b must also satisfy $\forall R.(\neg A \sqcup \neg B)$, and c, d were introduced as fillers of b for R, we get 2 more constraints $c \in (\neg A \sqcup \neg B)^{\mathcal{I}}$ and $d \in (\neg A \sqcup \neg B)^{\mathcal{I}}$.

7. Now, $c \in (\neg A \sqcup \neg B)^{\mathcal{I}}$ means $c \in (\neg A)^{\mathcal{I}}$ or $c \in (\neg B)^{\mathcal{I}}$. $c \in (\neg A)^{\mathcal{I}}$ clashes with the constraint $c \in A^{\mathcal{I}}$, implying that this choice leads to an obvious contradiction. Hence, we must choose $c \in (\neg B)^{\mathcal{I}}$. Similarly, we must choose $d \in (\neg A)^{\mathcal{I}}$ in order to satisfy the constrait $d \in (\neg A \sqcup \neg B)^{\mathcal{I}}$ without contradicting $d \in B^{\mathcal{I}}$.

8. Now, since we have satisfied all constraints without encountering an obvious contradiction, we can conclude that C is satisfiable. We have generated an interpretation $\mathcal{I}$ as proof of this fact: $\Delta^{\mathcal{I}} = \{b, c, d\}; R^{\mathcal{I}} = \{(b, c), (b, d)\}; A^{\mathcal{I}} = \{c\}$ and $B^{\mathcal{I}} = \{d\}$. This means that, for this interpretation, $b \in C^{\mathcal{I}}$ i.e. $b \in ((\exists R.A) \sqcap (\exists R.B))^{\mathcal{I}}$, but $b \notin (\exists R.(A \sqcap B))^{\mathcal{I}}$. This shows that $(\exists R.A) \sqcap (\exists R.B)$ is not subsumed by $\exists R.(A \sqcap B)$.

4 OWL - Web Ontology Language

OWL[4] is influenced from various earlier knowledge representation languages and paradigms. It is largely influenced by Frames, RDF/XML and RDFS, and largely inspired by OIL and DAML's idea of using DL. OWL is a layer over RDF and RDFS, and a revision of OIL+DAML. Different versions of OWL show different levels of semantic compatibility with RDF and RDFS. The next subsection gives a quick idea of the capabilities of RDF and RDFS, and OWL.

4.1 RDF, RDFS and OWL

Using RDF and RDFS, one can

- Declare classes like $Country$, $Person$, $Student$ and $Indian$.
- State that $Student$ is a subclass of $Person$.
- State that **India** and **China** are instances of class $Country$.
- Declare nationality as a property relating classes $Person$ and $Country$.
- State that age is a property with $Person$ as its domain and integer as its range.
- State that **Ram** is an instance of class $Indian$, and that his age has value **48**.

Using OWL, one can additionally

- State that $Country$ and $Person$ are disjoint classes.
- State that **India** and **China** are distinct individuals.
- Declare *hasCitizen* as the Inverse property of *nationality*.
- State that the class $Stateless$ is defined precisely as those members of class $Person$ that have no values for property *nationality*.
- State that class $MultipleNationals$ is defined precisely as those members of class $Person$ that have at least 2 values for property *nationality*.
- State that class $Indian$ is defined precisely as those members of class $Person$ that have **India** as the value of *nationality*.

4.2 Design influences on OWL

Use of DL formalized the semantics. It added automated reasoning to check knowledge base consistency and entailment. OWL design is based on SH family of DL. Abstract syntax of OWL is heavily influenced by Frames. Frames group together information about each class, thus making reading and understanding easier. A Class frame is equivalent to a DL axiom, and a Property frame is equivalent to a set of axioms asserting domain, range, transitivity etc. OWL was designed for maximum upward compatibility with RDF. The effort to make OWL semantics consistent with RDF semantics have led to 3 species of OWL - OWL Lite, OWL DL and OWL Full.

4.3 OWL Species

The species of OWL are all a result of different degrees of tradeoff between expressivity and efficient reasoning support. The characteristics of OWL DL, OWL Lite and OWL Full are listed below.

4.3.1 OWL DL

- Vocabulary partitioning: any resource is allowed to be only either a class, a datatype, a datatype property, an object property, an individual, a data value or part of the built-in vocabulary, and not more than one of these.
- Explicit typing: the partitioning mentioned above must be specified explicitly.
- Property separation: the set of object properties and datatype properties are disjoint.
- No transitive cardinality restrictions.
- Well formed RDF graphs a necessity, see [4].
- Inference is decidable, but takes non deterministic exponential time.
- Uses description logic SHOIN(D).

4.3.2 OWL Lite

- It is a syntactic subset of OWL DL.
- Unions and complements are not allowed.
- Individuals cannot be used in class axioms.
- Cardinalities can only be 0 or 1.
- Uses description logic SHIF(D).
- Inference is tractable, takes deterministic exponential time.
- FaCT and RACER reasoners some of the resoners that are available for OWL Lite.

4.3.3 OWL Full

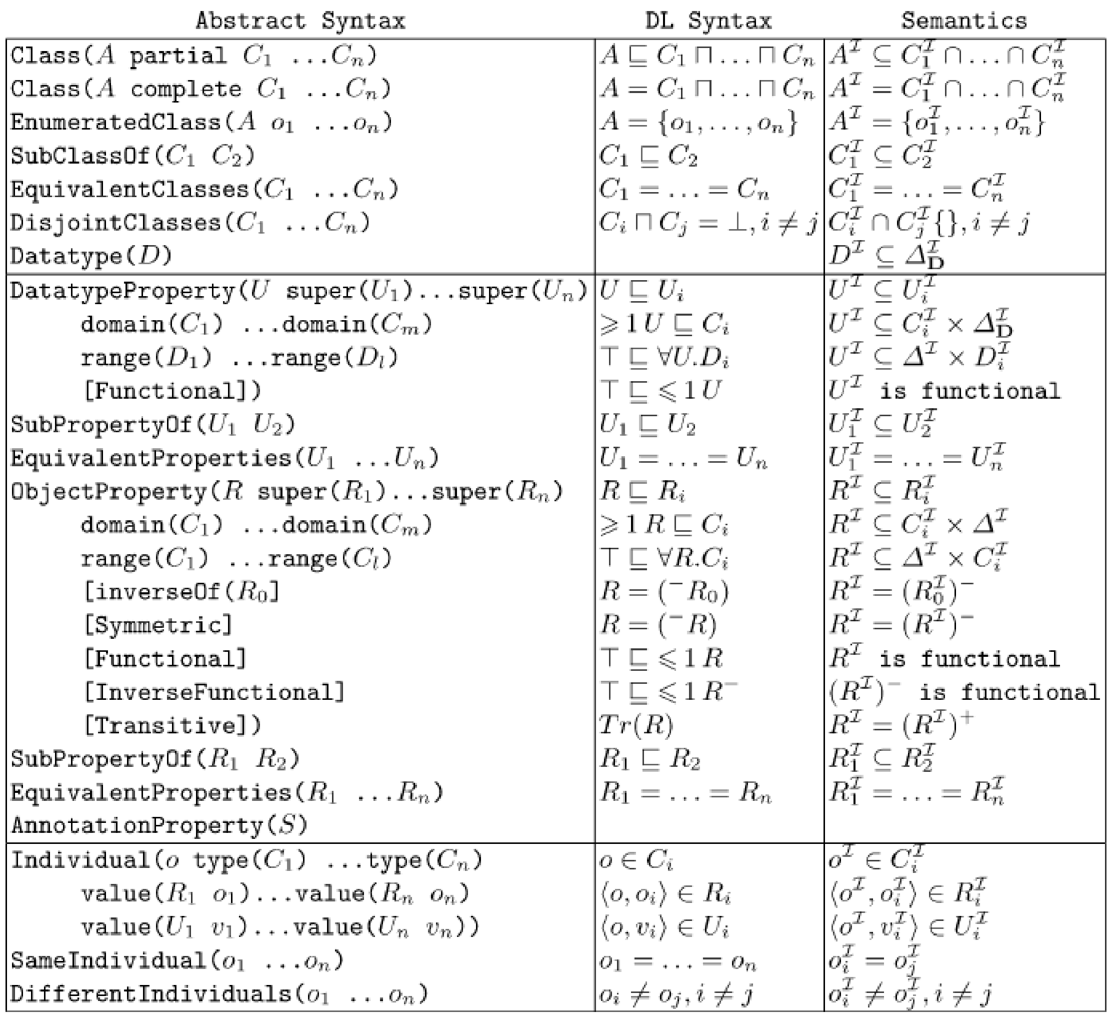
- It is a syntactic and semantic extension of RDF and RDFS.
- Reasoning is undecidable.
- It can include arbitrary RDF graphs.
- Goes well outside DL framework. Eg: You can even impose cardinality restriction on rdfs:subClassOf

4.4 OWL-DL Mapping

The OWL constructs are listed in the Appendix. Figures 6 and 7 show how OWL maps to DL.

Abstract Syntax	DL Syntax	Semantics
Descriptions (C)		
A (URI reference)	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
`owl:Thing`	$\top$	$\texttt{owl:Thing}^{\mathcal{I}} = \Delta^{\mathcal{I}}$
`owl:Nothing`	$\bot$	$\texttt{owl:Nothing}^{\mathcal{I}} = \{\}$
`intersectionOf(`$C_1\ C_2\ \ldots$`)`	$C_1 \sqcap C_2$	$(C_1 \sqcap D_1)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap D_2^{\mathcal{I}}$
`unionOf(`$C_1\ C_2\ \ldots$`)`	$C_1 \sqcup C_2$	$(C_1 \sqcup C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
`complementOf(`C`)`	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
`oneOf(`$o_1\ \ldots$`)`	$\{o_1, \ldots\}$	$\{o_1, \ldots\}^{\mathcal{I}} = \{o_1^{\mathcal{I}}, \ldots\}$
`restriction(`R `someValuesFrom(`C`))`	$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$
`restriction(`R `allValuesFrom(`C`))`	$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y.\langle x, y\rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
`restriction(`R `hasValue(`o`))`	$R : o$	$(\forall R.o)^{\mathcal{I}} = \{x \mid \langle x, o^{\mathcal{I}}\rangle \in R^{\mathcal{I}}\}$
`restriction(`R `minCardinality(`n`))`	$\geqslant n\, R$	$(\geqslant n\, R)^{\mathcal{I}} = \{x \mid \sharp(\{y.\langle x, y\rangle \in R^{\mathcal{I}}\}) \geqslant n\}$
`restriction(`R `minCardinality(`n`))`	$\leqslant n\, R$	$(\geqslant n\, R)^{\mathcal{I}} = \{x \mid \sharp(\{y.\langle x, y\rangle \in R^{\mathcal{I}}\}) \leqslant n\}$
`restriction(`U `someValuesFrom(`D`))`	$\exists U.D$	$(\exists U.D)^{\mathcal{I}} = \{x \mid \exists y.\langle x, y\rangle \in U^{\mathcal{I}} \text{ and } y \in D^{\mathbf{D}}\}$
`restriction(`U `allValuesFrom(`D`))`	$\forall U.D$	$(\forall U.D)^{\mathcal{I}} = \{x \mid \forall y.\langle x, y\rangle \in U^{\mathcal{I}} \rightarrow y \in D^{\mathbf{D}}\}$
`restriction(`U `hasValue(`v`))`	$U : v$	$(U : v)^{\mathcal{I}} = \{x \mid \langle x, v^{\mathcal{I}}\rangle \in U^{\mathcal{I}}\}$
`restriction(`U `minCardinality(`n`))`	$\geqslant n\, U$	$(\geqslant n\, U)^{\mathcal{I}} = \{x \mid \sharp(\{y.\langle x, y\rangle \in U^{\mathcal{I}}\}) \geqslant n\}$
`restriction(`U `maxCardinality(`n`))`	$\leqslant n\, U$	$(\leqslant n\, U)^{\mathcal{I}} = \{x \mid \sharp(\{y.\langle x, y\rangle \in U^{\mathcal{I}}\}) \leqslant n\}$
Data Ranges (D)		
D (URI reference)	D	$D^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}^{\mathcal{I}}$
`oneOf(`$v_1\ \ldots$`)`	$\{v_1, \ldots\}$	$\{v_1, \ldots\}^{\mathcal{I}} = \{v_1^{\mathcal{I}}, \ldots\}$
Object Properties (R)		
R (URI reference)	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
	R^-	$(R^-)^{\mathcal{I}} = (R^{\mathcal{I}})^-$
Datatype Properties (U)		
U (URI reference)	U	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_{\mathbf{D}}^{\mathcal{I}}$
Individuals (o)		
o (URI reference)	o	$o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
Data Values (v)		
v (RDF literal)	v	$v^{\mathcal{I}} = v^{\mathbf{D}}$

Figure 6: OWL DL descriptions, data ranges, properties, individuals and data values. (table adopted from [2])



Abstract Syntax	DL Syntax	Semantics
`Class(`A `partial` C_1 `...`C_n`)`	$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	$A^{\mathcal{I}} \subseteq C_1^{\mathcal{I}} \cap \ldots \cap C_n^{\mathcal{I}}$
`Class(`A `complete` C_1 `...`C_n`)`	$A = C_1 \sqcap \ldots \sqcap C_n$	$A^{\mathcal{I}} = C_1^{\mathcal{I}} \cap \ldots \cap C_n^{\mathcal{I}}$
`EnumeratedClass(`A o_1 `...`o_n`)`	$A = \{o_1, \ldots, o_n\}$	$A^{\mathcal{I}} = \{o_1^{\mathcal{I}}, \ldots, o_n^{\mathcal{I}}\}$
`SubClassOf(`C_1 C_2`)`	$C_1 \sqsubseteq C_2$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
`EquivalentClasses(`C_1 `...`C_n`)`	$C_1 = \ldots = C_n$	$C_1^{\mathcal{I}} = \ldots = C_n^{\mathcal{I}}$
`DisjointClasses(`C_1 `...`C_n`)`	$C_i \sqcap C_j = \bot, i \neq j$	$C_i^{\mathcal{I}} \cap C_j^{\mathcal{I}} \{\}, i \neq j$
`Datatype(`D`)`		$D^{\mathcal{I}} \subseteq \Delta_{\mathbf{D}}^{\mathcal{I}}$
`DatatypeProperty(`U `super(`U_1`)...super(`U_n`)`	$U \sqsubseteq U_i$	$U^{\mathcal{I}} \subseteq U_i^{\mathcal{I}}$
`domain(`C_1`) ...domain(`C_m`)`	$\geqslant 1\, U \sqsubseteq C_i$	$U^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta_{\mathbf{D}}^{\mathcal{I}}$
`range(`D_1`) ...range(`D_l`)`	$\top \sqsubseteq \forall U.D_i$	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times D_i^{\mathcal{I}}$
`[Functional])`	$\top \sqsubseteq \leqslant 1\, U$	$U^{\mathcal{I}}$ `is functional`
`SubPropertyOf(`U_1 U_2`)`	$U_1 \sqsubseteq U_2$	$U_1^{\mathcal{I}} \subseteq U_2^{\mathcal{I}}$
`EquivalentProperties(`U_1 `...`U_n`)`	$U_1 = \ldots = U_n$	$U_1^{\mathcal{I}} = \ldots = U_n^{\mathcal{I}}$
`ObjectProperty(`R `super(`R_1`)...super(`R_n`)`	$R \sqsubseteq R_i$	$R^{\mathcal{I}} \subseteq R_i^{\mathcal{I}}$
`domain(`C_1`) ...domain(`C_m`)`	$\geqslant 1\, R \sqsubseteq C_i$	$R^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
`range(`C_1`) ...range(`C_l`)`	$\top \sqsubseteq \forall R.C_i$	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C_i^{\mathcal{I}}$
`[inverseOf(`R_0`]`	$R = ({}^{-}R_0)$	$R^{\mathcal{I}} = (R_0^{\mathcal{I}})^{-}$
`[Symmetric]`	$R = ({}^{-}R)$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^{-}$
`[Functional]`	$\top \sqsubseteq \leqslant 1\, R$	$R^{\mathcal{I}}$ `is functional`
`[InverseFunctional]`	$\top \sqsubseteq \leqslant 1\, R^{-}$	$(R^{\mathcal{I}})^{-}$ `is functional`
`[Transitive])`	$Tr(R)$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^{+}$
`SubPropertyOf(`R_1 R_2`)`	$R_1 \sqsubseteq R_2$	$R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$
`EquivalentProperties(`R_1 `...`R_n`)`	$R_1 = \ldots = R_n$	$R_1^{\mathcal{I}} = \ldots = R_n^{\mathcal{I}}$
`AnnotationProperty(`S`)`		
`Individual(`o `type(`C_1`) ...type(`C_n`)`	$o \in C_i$	$o^{\mathcal{I}} \in C_i^{\mathcal{I}}$
`value(`R_1 o_1`)...value(`R_n o_n`)`	$\langle o, o_i \rangle \in R_i$	$\langle o^{\mathcal{I}}, o_i^{\mathcal{I}} \rangle \in R_i^{\mathcal{I}}$
`value(`U_1 v_1`)...value(`U_n v_n`))`	$\langle o, v_i \rangle \in U_i$	$\langle o^{\mathcal{I}}, v_i^{\mathcal{I}} \rangle \in U_i^{\mathcal{I}}$
`SameIndividual(`o_1 `...`o_n`)`	$o_1 = \ldots = o_n$	$o_i^{\mathcal{I}} = o_j^{\mathcal{I}}$
`DifferentIndividuals(`o_1 `...`o_n`)`	$o_i \neq o_j, i \neq j$	$o_i^{\mathcal{I}} \neq o_j^{\mathcal{I}}, i \neq j$

Figure 7: OWL DL axioms and facts. (table adopted from [2])

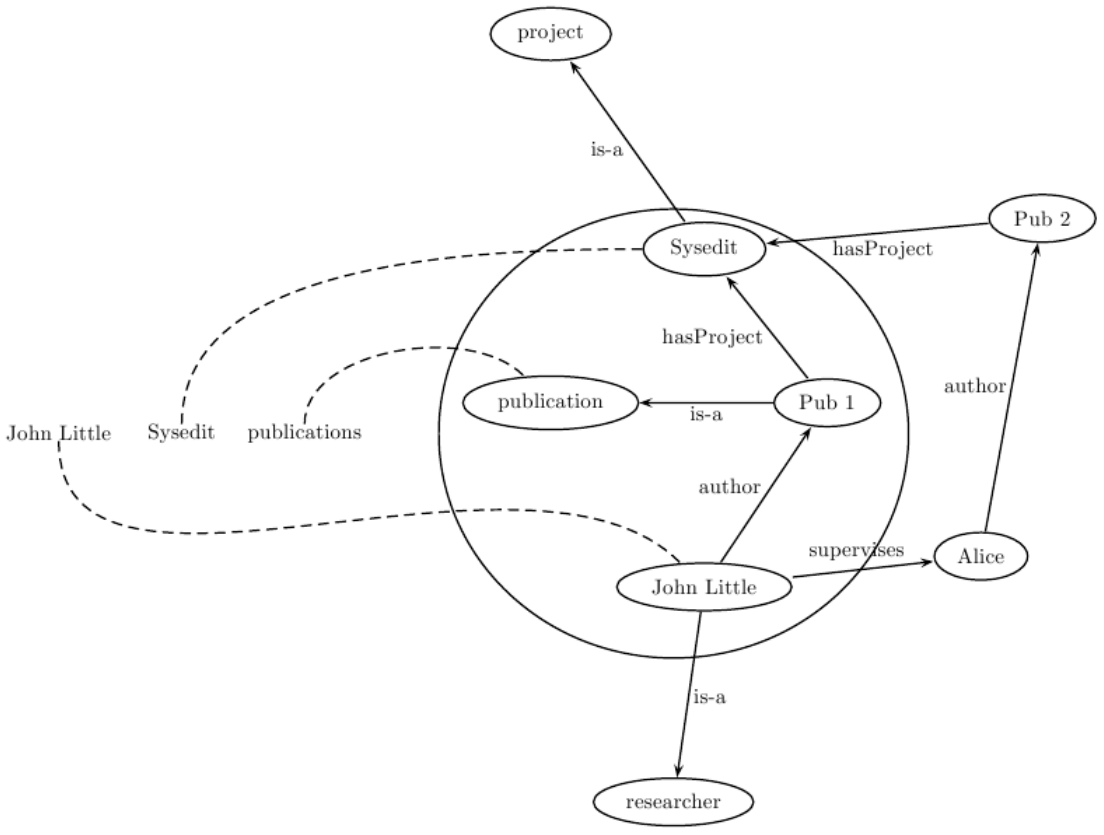


Figure 8: Querying

5 Querying

As illustrated in [6], this section intends to answer the question - how is the knowledgebase queried, or more specifically, how can keyword queries be converted to DL queries.

Suppose a user wants to retrieve all publications authored by John Little which are associated with project Sysedit. Say, the user uses a query $Q =$"John Little Sysedit publications".

The elements in the query are mapped to ontology elements John Little, Sysedit and publication. Exploration starts from the individual John Little and leads to the relations author and is-a. Assuming the exploration width is 2, elements Pub 1 and researcher are also reached from John Little. From Sysedit, we cover relations hasProject, is-a and elements project, Pub 1 and Pub 2. From publication, we cover is-a and reach Pub 1. Thus, step-by-step exploration builds a graph where all elements in the initial user query are present.

Next, the (possibly many) subgraphs which connect these elements are computed. These subgraphs correspond to the different questions the user possibly has. As highlighted in the circle in Figure 8, in this specific example there is only one such subgraph. However, in other scenarios, and in particular if the exploration range d is set higher, it is likely that several such subgraphs are obtained.

Finally, after discovering the relations author, is-a and hasProject between the user query words, the DL query

$$Q = \langle x, JohnLittle\rangle : name \wedge \langle x, y\rangle : author \wedge \langle y, z\rangle : hasProject \\ \wedge \langle z, Sysedit\rangle : name \wedge \langle y : publication\rangle$$

is formed. This DL query is then subjected to inference on the knowledge base, and all publications authored by John Little which are associated with project Sysedit are retrieved.

6 Conclusion

I have a dream for the Web in which computers become capable of analyzing all the data on the Web the content, links, and transactions between people and computers. A 'Semantic Web', which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The 'intelligent agents' people have touted for ages will finally materialize.
-Tim Berners-Lee, 1999

The semantic web has been an active area of research lately. At its core, the semantic web comprises a set of design principles, collaborative working groups, and a variety of enabling technologies. Some elements of the semantic web are expressed as prospective future possibilities that are yet to be implemented or realized.

References

[1] Grigoris Antoniou and Frank van Harmelen. Web ontology language: OWL. In *Handbook on Ontologies in Information Systems*, pages 76–92. Springer-Verlag, 2003.

[2] Franz Baader and Werner Nutt. Basic description logics. In *The description logic handbook: theory, implementation, and applications*, pages 43–95, New York, NY, USA, 2003. Cambridge University Press.

[3] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web: Scientific american. *Scientific American*, May 2001.

[4] Ian Horrocks, Peter F. Patel-schneider, and Frank Van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1:7–26, 2003.

[5] Ian Niles and Adam Pease. Towards a standard upper ontology. In *FOIS '01: Proceedings of the international conference on Formal Ontology in Information Systems*, pages 2–9, New York, NY, USA, 2001. ACM.

[6] Thanh Tran, Philipp Cimiano, Sebastian Rudolph, and Rudi Studer. Ontology-based interpretation of keywords for semantic search. In *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC2007), Busan, South Korea*, volume 4825 of *LNCS*, pages 519–532, Berlin, Heidelberg, November 2007. Springer Verlag.

[7] Wikipedia. Ontology — Wikipedia, the free encyclopedia, 2009. [Online; accessed 10-April-2009].

[8] Jidi Zhao. Introduction to description logic and ontology languages, 2008. Institute of Computer Technology, TU Vienna.

7 Appendix - OWL Constructs

As succinctly listed in [1], OWL uses the following constructs -

7.1 Class elements

Class elements are defined using owl:Class. We can define a class $associateProfessor$ as

```
<owl:Class rdf:ID="associateProfessor">
     <rdfs:subClassOf rdf:resource="#academicStaffMember"/>
</owl:Class>
```

We can further say that this class is disjoint from the $professor$ and $assistantProfessor$ classes using owl:disjointWith element.

```
<owl:Class rdf:about="associateProfessor">
     <owl:disjointWith rdf:resource="#professor"/>
     <owl:disjointWith rdf:resource="#assistantProfessor"/>
</owl:Class>
```

Equivalence of classes can be dened using a owl:equivalentClass element.

```
<owl:Class rdf:ID="faculty">
     <owl:equivalentClass rdf:resource="#academicStaffMember"/>
</owl:Class>
```

Further, there are 2 predefined classes, owl:Thing and owl:Nothing. Every class is a subclass of owl:Thing and a superclass of owl:Nothing

7.2 Property elements

There are 2 kinds of properties

- Object properties: relate objects to objects. We define that courses are taught by academic staff members.

  ```
  <owl:ObjectProperty rdf:ID="isTaughtBy">
        <owl:domain rdf:resource="#course"/>
        <owl:range rdf:resource="#academicStaffMember"/>
  </owl:ObjectProperty>
  ```

- Datatype properties: relate objects to datatype values. OWL uses XML Schema datatypes.

  ```
  <owl:DatatypeProperty rdf:ID="age">
       <rdfs:range rdf:resource="http://www.w3.org/2001/XLMSchema#nonNegativeInteger"/>
  </owl:DatatypeProperty>
  ```

OWL also allows one to define inverse properties. We can say that “teaches” is the inverse of “isTaughtBy” as under.

```
<owl:ObjectProperty rdf:ID="teaches">
    <rdfs:range rdf:resource="#course"/>
    <rdfs:domain rdf:resource="#academicStaffMember"/>
    <owl:inverseOf rdf:resource="#isTaughtBy"/>
</owl:ObjectProperty>
```

7.3 Value Restrictions

We can perform universal and existential quantifications, and have cardinality restrictions also. For example, to specify that first year courses must be taught by professors only, we can use owl:allValuesFrom restriction.

```
<owl:Class rdf:about="#firstYearCourse">
    <rdfs:subClassOf>
         <owl:Restriction>
               <owl:onProperty rdf:resource="#isTaughtBy"/>
               <owl:allValuesFrom rdf:resource="#Professor"/>
         </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

And we can declare that all academic staff members must teach at least one undergraduate course using owl:someValuesFrom restriction.

```
<owl:Class rdf:about="#academicStaffMember">
      <rdfs:subClassOf>
          <owl:Restriction>
                <owl:onProperty rdf:resource="#teaches"/>
                <owl:someValuesFrom rdf:resource="#undergraduateCourse"/>
          </owl:Restriction>
      </rdfs:subClassOf>
</owl:Class>
```

Further, we can state that every course must be taught by someone.

```
<owl:Class rdf:about="#course">
     <rdfs:subClassOf>
          <owl:Restriction>
                <owl:onProperty rdf:resource="#isTaughtBy"/>
                <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
                    1
                </owl:minCardinality>
          </owl:Restriction>
     </rdfs:subClassOf>
</owl:Class>
```

7.4 Boolean combinations - Union, Intersection

We can define concepts as boolean combinations of othere concepts. For example, we can say that people in the university are staff members and students.

```
<owl:Class rdf:ID="peopleInUniversity">
     <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#staffMember"/>
          <owl:Class rdf:about="#student"/>
     </owl:unionOf>
</owl:Class>
```

Further, we can say that administrative staff are those people who are neither faculty nor tech support staff.

```
<owl:Class rdf:ID="adminStaff">
    <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#staffMember"/>
          <owl:complementOf>
               <owl:unionOf rdf:parseType="Collection">
                      <owl:Class rdf:about="#faculty"/>
                      <owl:Class rdf:about="#techSupportStaff"/>
               </owl:unionOf>
           </owl:complementOf>
    </owl:intersectionOf>
</owl:Class>
```

7.5 Instances

We can say that "pb" is an academic staff member and teaches the course CS626 as under.

```
<academicStaffMember rdf:ID="pb">
     <uni:age rdf:datatype="&xsd;integer">39</uni:age>
</academicStaffMember>

<course rdf:about="CS626">
     <isTaughtBy rdf:resource="pb">
</course>
```

The complete OWL specification can be found at http://www.w3.org/2004/OWL/.