KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)
Coimbatore-21

LECTURE PLAN
COMPUTER APPLICATIONS

SUBJECT NAME: SEMANTIC WEB **SUBJECT CODE: 17CA505WB**
SEMESTER: V **LASS: III MCA**

SEMESTER-V

Scope: This course provides the methods to discover, classify and build ontology for searching. To build and implement a small ontology that is semantically descriptive of chosen problem domain. To implement applications that can access, use and manipulate the ontology.

Objectives:

- To represent data from a chosen problem in XML with appropriate semantic tags obtained or derived from the ontology.
- To understand the semantic relationships among these data elements using Resource.
- To design and implement a web services application that "discovers" the data and/or other Description Framework (RDF).web services via the semantic web Able to discover the capabilities and limitations of semantic web technology for many applications.

UNIT I

Introduction : Introduction to the Syntactic web and Semantic Web – Evolution of the Web – The visual and syntactic web – Levels of Semantics – Metadata for web information - The semantic web architecture and technologies –Contrasting Semantic with Conventional Technologies –Semantic Modeling - Potential of semantic web solutions and challenges of adoption

UNIT II

Ontological Engineering: Ontologies – Taxonomies –Topic Maps – Classifying Ontologies – Terminological aspects: concepts, terms, relations between them – Complex Objects –Subclasses and Sub-properties definitions – Upper Ontologies – Quality – Uses - Types of terminological resources for ontology building – Methods and methodologies for building ontologies – Multilingual Ontologies -Ontology Development process and Life cycle – Methods for Ontology Learning – Ontology Evolution – Versioning

UNIT III

Structuring And Describing Web Resources :Structured Web Documents - XML – Structuring – Namespaces – Addressing – Querying – Processing - RDF – RDF Data Model – Serialization Formats- RDF Vocabulary –Inferencing - RDFS – basic Idea – Classes – Properties- Utility Properties – RDFS Modeling for Combinations and Patterns- Transitivity

UNIT IV

Web Ontology Language :OWL – Sub-Languages – Basic Notions -Classes- Defining and Using Properties – Domain and Range – Describing Properties - Data Types – Counting and Sets- Negative Property Assertions – Advanced Class Description – Equivalence – Owl Logic.

UNIT V

Semantic Web Tools And Applications :Development Tools for Semantic Web – Jena Framework – SPARL –Querying semantic web - Semantic Wikis - Semantic Web Services – Modeling and aggregating social network data - Ontological representation of social relationships, Aggregating and reasoning with social network data Understand semantic web basics, architecture and technologies

TEXT BOOK:

1. Grigoris Antoniou, Frank van Harmelen. 3rd Edition 2012. A Semantic Web Primer.,MIT Press, USA

REFERENCES:

1. Liyang Yu, "A Developer's Guide to the Semantic Web", Springer, First Edition, 2011
2. John Hebeler, Matthew Fisher, Ryan Blace and Andrew Perez-Lopez, "Semantic Web Programming", Wiley, First Edition, 2009.
3. Robert M. Colomb, "Ontology and the Semantic Web", Volume 156 Frontiers in Artificial Intelligence and Applications (Frontier in Artificial Intelligence and Applications), IOS Press, 2007.
4. Dean Allemang and James Hendler, "Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL, Morgan Kaufmann", Second Edition, 2011.
5. Karin Breitman, Marco Antonio Casanova and Walt Truszkowski, "Semantic Web: Concepts, Technologies and Applications (NASA Monographs in Systems and Software Engineering)", Springer, Softcover, 2010.

CIA	Max.Marks(50)
Part A	Objective type questions - 20 x 1 = 20 Marks
Part B	Answer all the questions Either/Or - 3 x 10 = 30 Marks

ESE	Max.Marks(60)
Part A	Objective type questions -20 x 1 = 20 Marks
Part B	Answer all the questions Either/Or - 5 x 6 = 30 Marks
Part C	Answer all the questions Compulsory-1 x 10 = 10Marks

KARPAGAM ACADEMY OF HIGHER EDUCATION

CLASS: III MCA COURSE NAME: SEMANTIC WEB

COURSE CODE: 17CAP505W UNIT: I BATCH: 2017-2020

UNIT-1

SYLLABUS

Introduction : Introduction to the Syntactic web and Semantic Web – Evolution of the Web – The visual and syntactic web – Levels of Semantics – Metadata for web information - The semantic web architecture and technologies –Contrasting Semantic with Conventional Technologies –Semantic Modeling - Potential of semantic web solutions and challenges of adoption

INTRODUCTION TO SEMANTIC WEB:

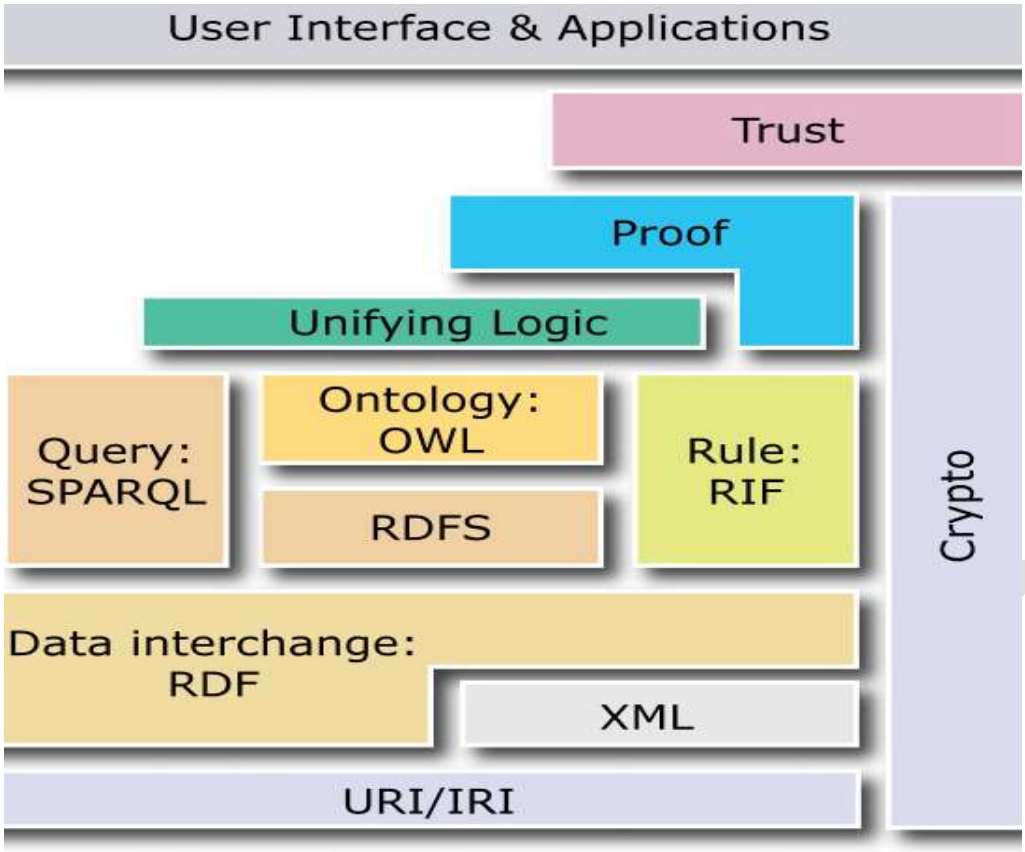
"The Semantic Web is an extension of the current web in which information in given well-define meaning, better enabling computers and people to work in co-operation."Currently most of the Web contentis suitable for human use._ Typical uses of the Web today are information seeking, publishing, and using, searching for people and products, shopping, reviewing catalogues, etc.

_Dynamic pages generated based on information from databases but without original informationstructure found in databases. The Web search results are high recall, low precision. Results are highly sensitive to vocabulary. Results are single Web pages. Most of the publishing contents are not structured to allow logical reasoning and query answering.

Organising knowledge in conceptual spaces according to its meaning _ Enabling automated tools to check for inconsistencies and extracting new knowledge. Replacing query-based search with query answering.

_ Defining who may view certain parts of informationdefine exhaustive description frameworks for describing Web Services and related aspects (Web Service Description Ontologies).

_ support ontologies as underlying data model to allow machine supported data interpretation (Semantic Web aspect) definsemantically driven technologies for automation of the Web Service usage process (Web Service aspect)



Semantic web version

Metadata and the Web

When the first edition of this book was published in 1998, the term *metadata* was comparatively esoteric, having originated in the information science and geospatial data communities before being co-opted and partially redefined by the library, archive, and museum information communities at the end of the twentieth century. Today, nearly a decade later, a Google search on "metadata" yields about 58 million results (see Web Search Engines sidebar). Metadata has quietly hit the big time; it is now a consumer commodity.

For example, almost all consumer-level digital cameras capture and embed Exchangeable Image File Format (EXIF)[1] metadata in digital images, and files created using Adobe's Creative Suite

of software tools (e.g. Photoshop) contain embedded Extensible Metadata Platform (XMP)[2] metadata.

As the term *metadata* has been increasingly adopted and co-opted by more diverse audiences, the definition of what constitutes metadata has grown in scope to include almost anything that describes anything else. The standard concise definition of metadata is "data about data," a relationship that is frequently illustrated using the metaphor of
library card catalog.

The first few lines of the following Wikipedia entry for *metadata* are typical: Metadata (Greek: meta- + Latin: data "information"), literally "data about data," are information about another set of data. A common example is a library catalog card, which contains data about the contents and location of a book: They are data about the data in the book referred to by the card. The library catalog card metaphor is pedagogically useful becauseit is nonthreatening.

Most people are familiar with the concept of a card catalog as a simple tool to help readers find the books they are looking for and to help librarians manage a library's collection as a whole.

Although the Netcraft Web hosts survey clearly demonstrates the continuing upward trend in the growth of the Web, it does not tell the whole story because it does not address how many Web sites are hosted on each server or how many accessible pages are contained in each site.

Semantic web Technology, Layered Architecture

The World Wide Web has changed the way people communicate with each other and the way business is conducted. It lies at the heart of a revolution that is currently transforming the developed world toward a knowledge economy and, more broadly speaking, to a knowledge society.

This development has also changed the way we think of computers. Originally they were used for computing numerical calculations. Currently their predominant use is for information processing, typical applications being databases, text processing and games.

At present there is a transition focus towards the view of computers as entry points to the information high ways. Most of today‘s web content is suitable for human consumption. Even web content that is generated automatically from databases is usually presented without the original structural information found in databases.

Typical uses of the web today involve people‘s seeking and making use of information, searching for and getting in touch with other people, reviewing catalogs of online stores and ordering products by filling out forms, and viewing material.

These activities are not particularly well supported by software tools. Apart from the existence of links that establish connections between documents, the main valuable, indeed indispensable, tools are search engines.

Keyword-based search engines, such as AltaVista, Yahoo, and Google, are the main tools for using today‘s web. It is clear that the web would not have been the huge success it was, were it not for search engines. However, there are serious problems associated with their use.

The evolutions of Web:

The Semantic Web was made through incremental changes, by bringing machine-readable descriptions to the data and documents already on the Web. The following figure illustrates the various developed technologies that made the concept of the Semantic Web possible.

Data breaks down into three broad categories : unstructured, semistructured, and structured.

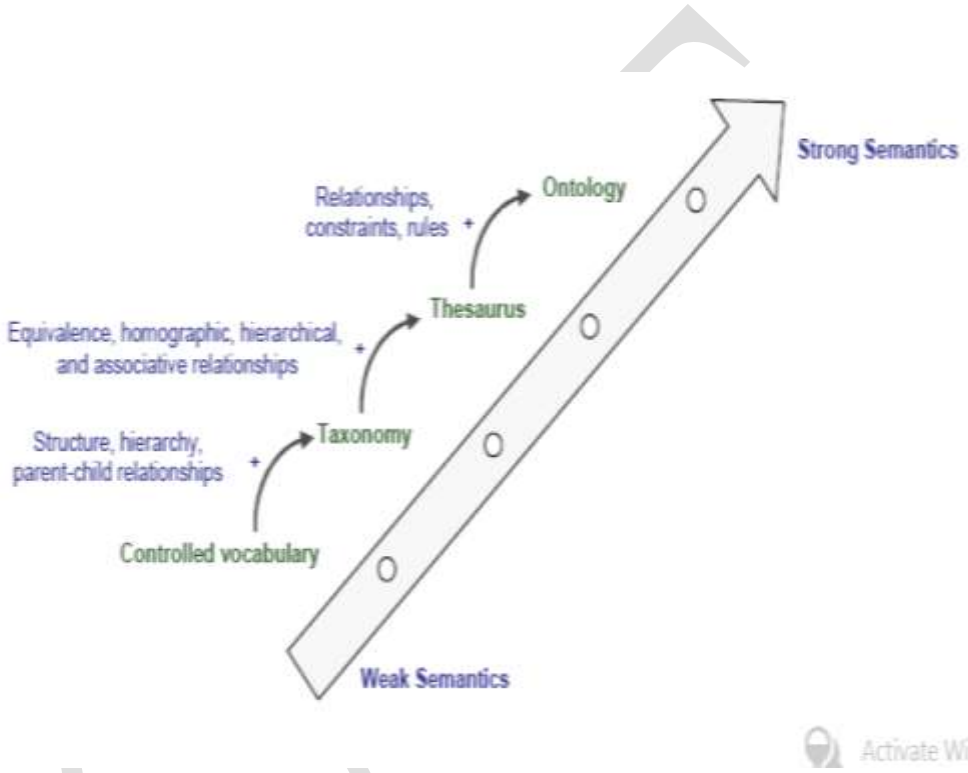
A) Unstructured Data: Unstructured data is what we find in text, files, video, e-mails, reports, PowerPoint presentations, voice mail, office memos, and images. Data can be of any type and does not necessarily follow any format, rules, or sequence. For example, the data present on HTML Web pages is unstructured and irregular. Unstructured data does not readily fit into structured databases except as binary large objects (BLOBs-binary large objects).

B) Semi-structured data: Semi-structured data lie somewhere in between unstructured and structured data. Semi structured data are data that have some structure, but are not rigidly structured. This type of data includes unstructured components arranged according to some predetermined structure. Semi structured data can be specified in such a way that it can be queried using general-purpose mechanisms. Semi structured data are organized into entities. Similar entities are grouped together, but entities in the same group may not have the same attributes.

C) Structured data: In contrast, structured data *are very rigid* and describe objects using strongly typed attributes, which are organized as records or tuples. *All records have the same fields.* Data are organized in entities and similar entities are grouped together using relations or classes. Entities in the same group have the same attributes. The descriptions for all the entities in a schema have the same defined format, predefined length, and follow the same order.

Levels of Semantics:

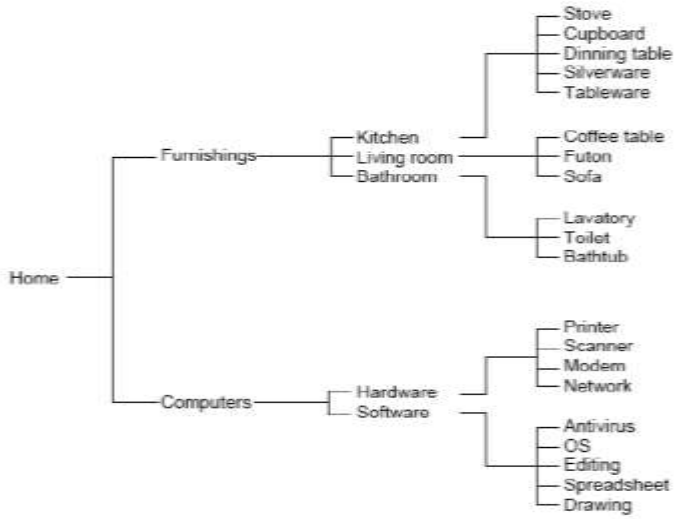
Semantics is the study of the meaning of signs, such as terms or words. Depending on the approaches, models, or methods used to add semantics to terms, different degrees of semantics can be achieved. The levels of semantics is shown in the following figure.



A) Controlled vocabulary: A controlled vocabulary is a list of terms (e.g., words, phrases, or notations) that have been enumerated explicitly. All terms in a controlled vocabulary should have an unambiguous, non-redundant definition. For example, Amazon.com has the following (Table 1) controlled vocabulary which can be selected by the user to search for products.

B) Taxonomy: A *taxonomy*is a subject-based classification that arranges the terms in a controlled vocabulary into a hierarchy without doing anything further. The taxonomy arrangement in a home is shown in Figure.

Figure 5. Example of a taxonomy



C) Thesaurus : A thesaurus is a networked collection of controlled vocabulary terms with conceptual relationships between terms. A thesaurus is an extension of a taxonomy by allowing terms to be arranged in a hierarchy and also allowing other statements and relationships to be made about the terms. The following table shows the semantic relationships of a Thesaurus with suitable example.

SEMANTIC RELATION	DEFINITION	EXAMPLE
Synonym Similar to Equivalent Used for	A term *X* has nearly the same meaning as a term *Y*.	"Report" is a synonym for "document."
Homonym Spelled the same Homographic	A term *X* is spelled the same way as a term *Y*, which has a different meaning.	The "tank," which is a military vehicle, is a homonym for the "tank," which is a receptacle for holding liquids.
Broader Than (Hierarchic: parent of)	A term *X* is broader in meaning than a term *Y*.	"Organization" has a broader meaning than "financial institution."
Narrower Than (Hierarchic: child of)	A term *X* is narrower in meaning than a term *Y*.	"Financial institution" has a narrower meaning than "organization."
Associated Associative Related	A term *X* is associated with a term *Y*, i.e., there is some unspecified relationship between the two.	A "nail" is associated with a "hammer."

C) Ontology: Ontologies are similar to taxonomies but use richer semantic relationships among terms and attributes, as well as strict rules about how to specify terms and relationships. Ontologies have generally been associated with logical inferencing and recently have begun to be applied to the semantic Web. An ontology is a shared conceptualization of the world. Ontologies consist of definitional aspects such as high-level schemas and assertional aspects such as entities, attributes, interrelationships between entities, domain vocabulary and factual knowledge—all connected in a semantic manner.

Semantic Web : Layered Architecture

Semantic Web is the new generation Web that tries to represent information such that it can be used by machines, not just for display purposes, but for automation, integration, and reuse across applications (Berners-Lee 2000).

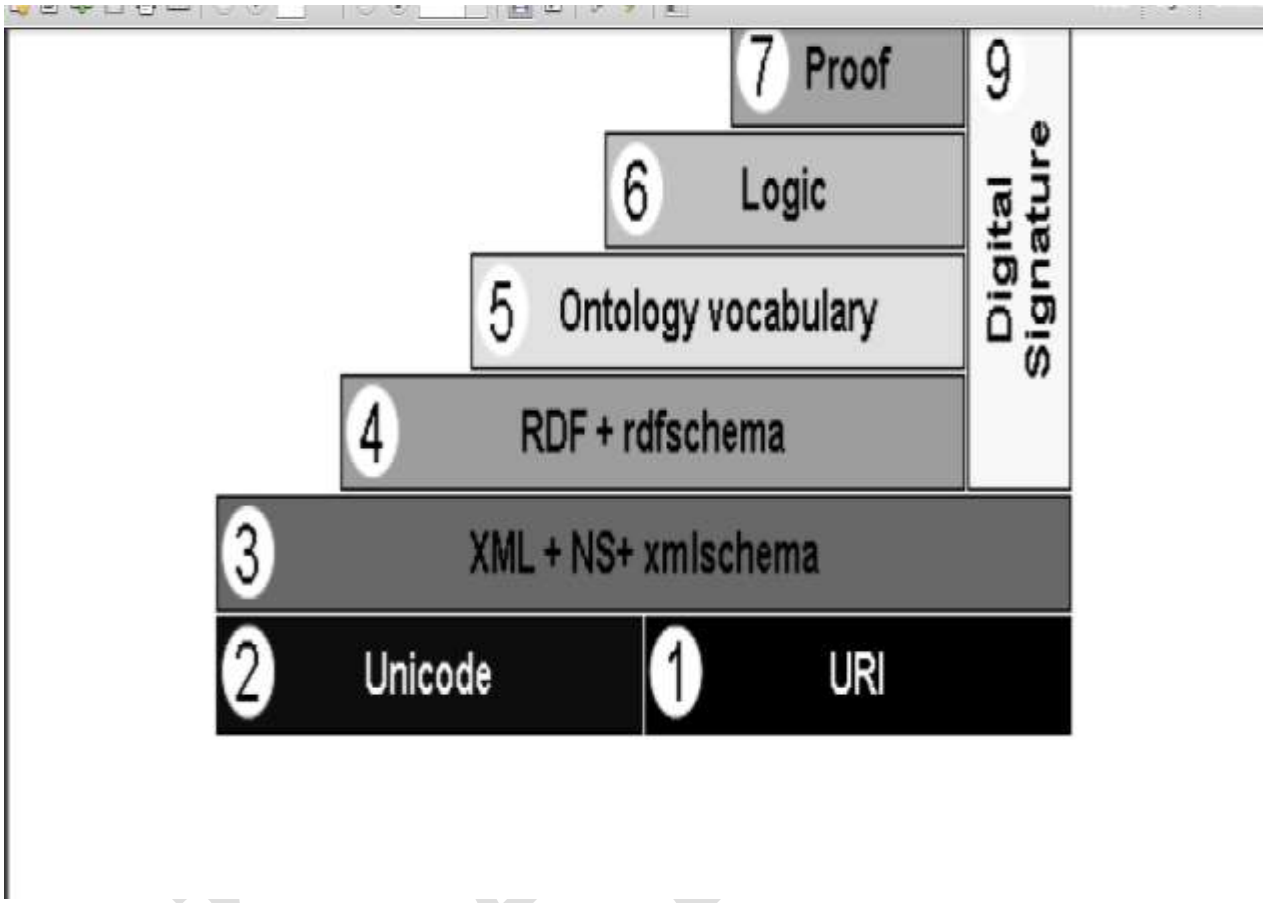
Furthermore, semantic Web is about explicitly declaring the knowledge embedded in many Web based applications, integrating information in an intelligent way, providing semantic based access to the Internet, and extracting information from texts.

Traditionally, HTML provides the standard of structured document published on the Internet. Though the simplicity of HTML promotes the growth of the Web, it seriously hampered advanced applications such as processing, understanding and semantic interoperability of information contained in several documents.

Semantic Web is the new generation Web which makes possible to express information in precise, machine-interpretable form. It enables intelligent services such as information brokers, search agents and information filters, and also offers greater functionality and interoperability.

Semantic Web promotes Web based applications with both semantic and syntactic interoperability. The explicit representation of meta-information, accompanied by domain theories (i.e. ontologies), will enable a Web to provide a qualitatively new level of service.

This process may ultimately create extremely knowledgeable systems with various specialized reasoning services. The architecture of semantic Web (W3C) is shown in Figure. The semantic Web technologies offer a new approach to managing information and processes, the fundamental principle of which is the creation and use of semantic metadata.



(i) URI

A universal resource identifier (URI) is a formatted string that serves as a means of identifying abstract or physical resource. A URI can be further classified as a locator, a name, or both. Uniform resource locator (URL) refers to the subset of URI that identifies resources via a representation of their primary access mechanism.

(ii) Unicode

Unicode provides a unique number for every character, independently of the underlying platform, program, or language. Before the creation of Unicode , there were various different

encoding systems. The diverse encoding made the manipulation of data complex. Any given computer needed to support many different encodings.

There was always the risk of encoding conflict, since two encodings could use the same number for two different characters, or use different numbers for the same character. Examples of older and well known encoding systems include ASCII and EBCDIC.

(iii)XML and XML Namespace

XML (eXtensible markup language) with XML namespace and XML schema definitions makes sure that there is a common syntax used in the semantic Web. XML namespaces allow specifying different markup vocabularies in one XML document. XML schema serves for expressing schema definition of a particular XML document. When it comes to semantic interoperability, however, XML has disadvantages.

(iv) RDF and RDF Schema

On top of XML is the Resource Description Framework (RDF), for representing information about resources in a graph form. RDF is based on triples O-A-V that form a graph data with a relation among object (a resource), an attribute (a property), and a value (a resource).

RDF Schema (RDFS) defines the vocabulary of RDF model.

It provides a mechanism to describe domain-specific properties and classes of resources to which those properties can be applied, using a set of basic modeling primitives (class, subclass-of, property, subproperty-of, domain, range, type). However, RDFS is rather simple and it still does not provide exact semantics of a domain.

(v) Ontology

Ontology comprises a set of knowledge terms, including the vocabulary, the semantic interconnections, simple rules of inference and logic for some particular topic. Ontologies applied to the Web are creating the semantic Web.

Ontologies facilitate knowledge sharing and provide reusable Web contents, Web services, and applications. Few of the ontology languages are DAML (DARPA Agent Markup Language), OIL (Ontology Interference Layer) and OWL (Web Ontology Language). OWL is developed starting from description logic and DAML+OIL.

(vi) Logic, Proof, Trust and Digital Signature

The logic layer is used to enhance the ontology language further and to allow the writing of application-specific declarative knowledge.

The proof layer involves the actual deductive process as well as the representation of proofs in Web languages and proof validation. Finally, the Trust layer will emerge through the use of digital signatures and other kinds of knowledge, based on recommendations by trusted agents or on rating and certification agencies and consumer bodies.

Three levels of meaning

■ **Assertions** (truth-conditions, entailment)

■ **Presuppositions** The requirements that the context must satisfy for the utterance to be interpretable at all

■ **Conversational Implicatures** Inferences that arise from observing or flouting the cooperative principle and conversational maxims

Presuppositions

■ **Basic idea:** A presupposition of a statement is a proposition thatmust be true in order for the statement to beinterpretable (to make sense) in the first place.

■ **Slightly different view:** A presupposition is an implicit assumption about theworld whose truth is taken for granted by the speaker.

Challenges in the Adoption and Diffusion of Web Services

During the last years some analysts have tried to sell an envisioned almost perfect IT Information Technology world though the use of web services. A company might be able to assemble a whole business by piecing together web services created and maintained by other companies, and listed in public online directories.

For example, "a company looking for a way to check the credit histories of its customers could have its order-processing software automatically scan the Web to find companies that offer a credit-checking web service, figure out which company offers the best deal, negotiate it and then hook up to that company's web service on the fly-even if the two companies never did business together before".

Examples of current and possible applications of web services and related challenges are used to illustrate the ideas, but they can not be taken for granted. When confronted with a new business phenomenon many times companies look for marketplace outcomes for guidance. "But, in the early stages of the rollout of any important technology, market signal can be unreliable. New technologies trigger rampant experimentation, by both companies and customers, and the experimentation is often economically unsustainable. As a result, market behavior is distorted and must be interpreted with caution"

Introduction to Web Services

The work is intended to evaluate the potential of web services in generating value to the business, and not dive in the technical side of the technology. But to do so, it is necessary to understand some technical issues and evolution of the web services' components, structure, and

standards, because the development and deployment of web services are in great part related to them.

The web services concept is still under development and it is possible to find a lot of different definitions to them. Vendors try to establish definitions that best match their capabilities, organizations responsible for the development of standards try to establish definitions that give them power over the greater number of standards, and many others entities have the definitive definition to web services. Furthermore, there is a lot of discussion to analyze whether the current implementations carried out by companies and denominated web services, are really web services or not.

Analyzing diverse cases and comparing with accepted definitions of web services, it is possible to realize that most of them are implementations of standards that are supposed to compose web services, but not exactly web services. This work will not delve in such discussion, and instead will analyze the potential generation of value of web services or their components, as they have been deployed so far.

Challenges for Adoption

The same vendors that some time ago painted the scenario of web services as something quite simple and cheap ("integration almost for free") now include web services consultancy in their portfolios, because since web services evolved, their planning, deploying, and managing are getting more and more complex.

"Managing and tracking web services across an enterprise and ensuring interoperability, security, and performance requires a new order of architectural discipline" [58]. Companies as IBM, Accenture, Deloitte Consulting, EDS, and Hewlett Packard's are offering web services solutions as part of their overall IT**services portfolio.**

Many surveys show the tendency of companies to adopt web services, but the figures presented should not be taken for granted, for a lot of reasons such as: responders can fear the seeming out of the bandwagon, have the desire to show knowledge, have their sensitivity altered

as to the price or to the immaturity of the technology, and the surveys many times are not answered by the people responsible for making decisions in the companies.

Furthermore, in most of the surveys it is not possible to know either the number of answers analyzed or the characteristics of the companies involved. But, anyway, it is possible to try to analyze pointed reasons for adopting or not web services and put them together with possible lacking reasons. For instance, one survey shows that concerns about security are higher in those institutions that have already experimented web services.

While in one hand it can be simply a result of the sample adopted, it can indicate that the hype around the technology can lead some companies that have not implemented web services yet, believe in a more mature product, whereas those experiencing them can face an unexpected level of difficulty.

Semantic Issues

It is important to align data definitions, jargon, and vocabulary words within and across organizations ,solving concerns about the semantic and context of the information. Such definitions are critical to theinteroperability of systems and the implementation of web services. Simple divergences as to whether the“number of employees” refers to temporary or permanent ones, as well as whether the “ROI - Return onInvestment” is annualized or the amount until that date demand human intervention to make systemsinteroperable or time-consuming data consolidation. Different systems show different specifications as tounits of measure (currency for example), different time references, and different definitions of terms,

demanding a time-consuming analysis, being prone to error consolidation. The human being is normally ableto distinguish among the same term to different meanings, but the systems usually are not. For example, onesystem can have a list of addresses used to bill clients. This list can be sent to another company or system, tomake deliveries to these clients. Some of the addresses, though, can be post-office boxes, but its definition“address” does not allow the other systems to

know such thing, and deliveries can be made erroneously topost-office boxes instead of home addresses.

POSSIBLE QUESTIONS

UNIT I

PART-A (20 Marks)

(Q.No 1 to 20 Online Examination)

PART-B (6 Marks)

Part –C (10 Marks)

KARPAGAM ACADEMY OF HIGHER EDUCATION
COIMBATORE - 21

DEPARTMENT OF COMPUTER APPLICATIONS

CLASS : III MCA **BATCH : 2016-2019**

SUBJECT: SEMANTIC WEB **SUBJECT CODE: 17CAP505W**

	Questions	OPTION 1	OPTION 2	OPTION 3	OPTION 4	ANSWER
1	The Semantic Web	a Web without a meaning	a Web with a meaning	a Web without any reason	a web	a Web with a meaning
2	If HTML and the Web made all the online documents look like one huge	book	table	website	Internet	book
3	RDF, schema, and inference languages will make all the data in the world look like one huge	table	book	database	website	database
4	The semantic of something is the	structure of something	meaning of something	appearance of something	None of these	meaning of something
5	The Semantic Web is a web that is able to describe things in a way that computers can	convert	not understand	understand	compile	understand
6	Statements are built with	without any rules	syntax rules	None of these	semantic rules	syntax rules
7	The syntax of a language defines the rules for building the language	statements	objects	classes	functions	statements
8	The Semantic Web is not_____between web pages.	appearance	text	about links	String	about links
9	The Semantic Web describes the relationships between things (like A is a part of B and Y is a member of Z) and the properties of things (like size, weight, age, and price)	True	FALSE	Not always true	partial True	True
10	RDF is a language	Resource	markup language	programming language	an tool	markup
11	If information about music, cars, tickets were stored in RDF files, intelligent web applications could collect information from many different sources, combine information, and present it to users in a meaningful way.	FALSE	TRUE	Not always true	partial True	TRUE
12	The Semantic Web is not a very fast growing	tool	website	software	technology	technology
13	RDF was developed by people with academic background in	programming languages	logic and artificial intelligence.	None of these	web designing	logic and artificial intelligence
14	One fast growing language for building semantic web applications is	RSS	RDF	XSL	SOAP	RSS
15	_____is data about web data - or metadata	RDF	RSS	SOAP	XSL	RDF
16	Often RDF files describe other	RSS files	SOAP files	RDF files	XSL files	RDF files
17	The semantic web will not be searchable in	text	free text	odd text	None of these	free text
18	To search (or to access) the semantic web, we will need some the_____to help us.	software	hardware	website	tool	software
19	What is RDF ?	Resource Data Framework	Record Description Framework	Resource Description Framework	Record Data Framework	Resource Description Framework
20	web project which allows exchange of informaion among different computers on web is classified as	segregate web	conceptual web	semantic web	aggrigate web	semantic web
21	The semantic web uses RDF to describe	web resource	web contents	web controls	web framework	web resource
22	concept which describe domain of knowledge is considered as	morphology	ontology	anthropology	terminology	ontology
23	_____allows the representation of information that is also machine accessible.	OWL	HTML	XML	DHTML	XML
24	_____is not a function of XML	structure information	style information	transport information	store information	style information
25	XML deals with storage and_____transport of data	minity	transport	design	filter	transport
26	Fundamentals concepts of RDF is	resources	variables	constant	identifies	resources
27	Special kind of resource that describe relations between resources.	statements	triple	facets	property	property
28	An_____is a model of the common objects that are generally applicable across a wide range of domain ontology.	upper ontology	left ontology	right ontology	lower ontology	upper ontology
29	RDF Schema is a_____ontology language.	non-primitive	derived	primitive	user-defined	primitive

UNIT:II

SYLLABUS

Ontological Engineering: Ontologies – Taxonomies –Topic Maps – Classifying Ontologies – Terminological aspects: concepts, terms, relations between them – Complex Objects –Subclasses and Sub-properties definitions – Upper Ontologies – Quality – Uses - Types of terminological resources for ontology building – Methods and methodologies for building ontologies – Multilingual Ontologies -Ontology Development process and Life cycle – Methods for Ontology Learning – Ontology Evolution – Versioning

Ontological Engineering:

Ontologies:

Ontology-explicit formal specifications of the terms in the domain and relations among them (Gruber 1993).

Ontology is a key technique with which to annotate semantics and provide a common, comprehensible foundation for resources on the Semantic.

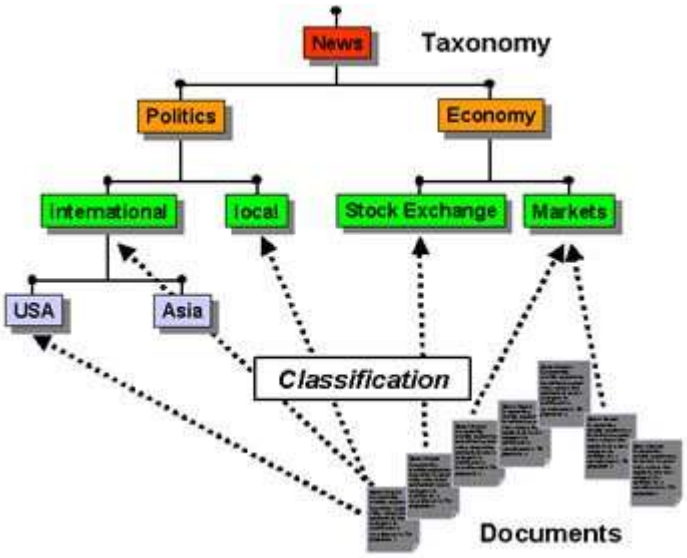
An ontology is a formal explicit description of concepts in a domain of discourse (classes (sometimes called concepts)), properties of each concept describing various features and attributes of the concept (slots (sometimes called roles or properties)), and restrictions on slots (facets (sometimes called role restrictions)). An ontology together with a set of individual instances of classes constitutes a knowledge base.

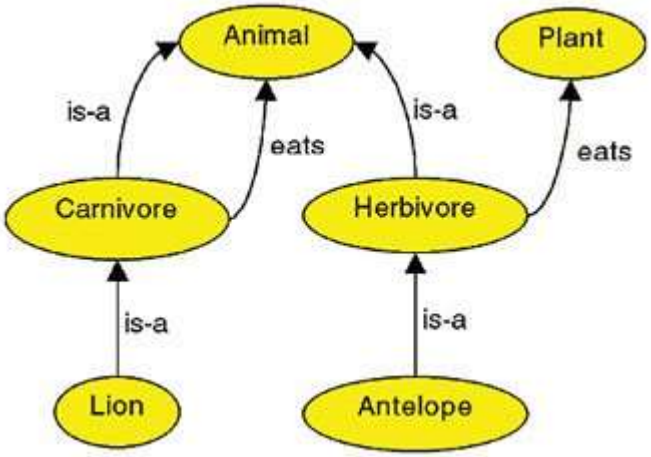
Taxonomy:

Taxonomy is usually only a hierarchy of concepts (i.e. the only relation between the concepts is parent/child, or subClass/superClass, or broader/narrower).

Taxonomy takes into consideration one type of relationship, whereas ontology takes into account many different complex relationships between the concepts.

The primary purpose of taxonomy and ontology is to help categorize and classify. Taxonomies are a system of classifying objects. Most taxonomies have a hierarchical structure, but it is not a requisite. It can be said that a taxonomy is simple hierarchical arrangement of entities. It utilizes a parent-child relation in its classification.





The main difference between Taxonomy and Ontology is that taxonomy is simpler in nature than ontology. Taxonomy takes into consideration one type of relationship, whereas ontology takes into account many different complex relationships between the concepts

Topic Maps:

Topic map is relatively a new entrant to this information space. Topic map standard describes how complex relationships between abstract concepts and real world resources can be represented using XML syntax.

A topic map is a standard for the representation and interchange of knowledge, with an emphasis on the findability of information. Topic maps were originally developed in the late 1990s as a way to represent back-of-the-book index structures so that multiple indexes from different sources could be merged. However, the developers quickly realized that with a little additional generalization, they could create a meta-model with potentially far wider application. The ISO standard is formally known as ISO/IEC 13250:2003.

A topic map represents information using

- topics, representing any concept, from people, countries, and organizations to software modules, individual files, and events,
- associations, representing hypergraph relationships between *topics*, and
- occurrences, representing information resources relevant to a particular *topic*.

Topic maps are similar to concept maps and mind maps in many respects, though only topic maps are ISO standards. Topic maps are a form of semantic web technology similar to RDF.

Terminological Aspects:

Terminology is the study of terms and their use. Terms are words and compound words or multi-word expressions that in specific contexts are given specific meanings—these may deviate from the meanings the same words have in other contexts and in everyday language. Terminology is a discipline that studies, among other things, the development of such terms and their interrelationships within a specialized domain. Terminology differs from lexicography, as it involves the study of concepts, conceptual systems and their labels (*terms*), whereas lexicography studies words and their meanings.

Terminology is a discipline that systematically studies the "labelling or designating of concepts" particular to one or more subject fields or domains of human activity. It does this through the research and analysis of terms in context for the purpose of documenting and promoting consistent usage. Terminology can be limited to one or more languages (for example,

"multilingual terminology" and "bilingual terminology"), or may have an interdisciplinarity focus on the use of terms in different fields.

Overview:

The discipline of terminology consists primarily of the following aspects:

- analyzing the concepts and concept structures used in a field or domain of activity
- identifying the terms assigned to the concepts
- in the case of bilingual or multilingual terminology, establishing correspondences between terms in the various languages
- compiling the terminology, on paper or in databases
- managing terminology databases
- creating new terms, as required

Types of terminology:

A distinction is made between two types of terminology work:

- Ad hoc work on terminology, which deals with a single term or a limited number of terms
- Systematic collection of terminology, which deals with all the terms in a specific subject field or domain of activity, often by creating a structured ontology of the terms within that domain and their interrelationships.

Ad hoc terminology is prevalent in the translation profession, where a translation for a specific term (or group of terms) is required quickly to solve a particular translation problem.

Ontology Development process and Life cycle:

Methodologies broadly divide into those that are stage-based (e.g. TOVE) and those that rely on iterative evolving prototypes (e.g. Methontology). These are in fact complementary techniques. Most distinguish between an informal stage, where the ontology is sketched out using either natural language descriptions or some diagram technique, and a formal stage where the ontology is encoded in a formal knowledge representation language, that is machine computable. As an ontology should ideally be communicated to people and unambiguously interpreted by software, the informal representation helps the former and the formal the latter.

Figures 1 and 2 represents a skeletal methodology and life-cycle for building ontologies, inspired by the software engineering V-process model . The left side of the V charts the processes in building an ontology and the right side charts the guidelines, principles and evaluation used to `quality assure' the ontology. The overall process, however, moves through a life-cycle, as depicted in Figure 2.

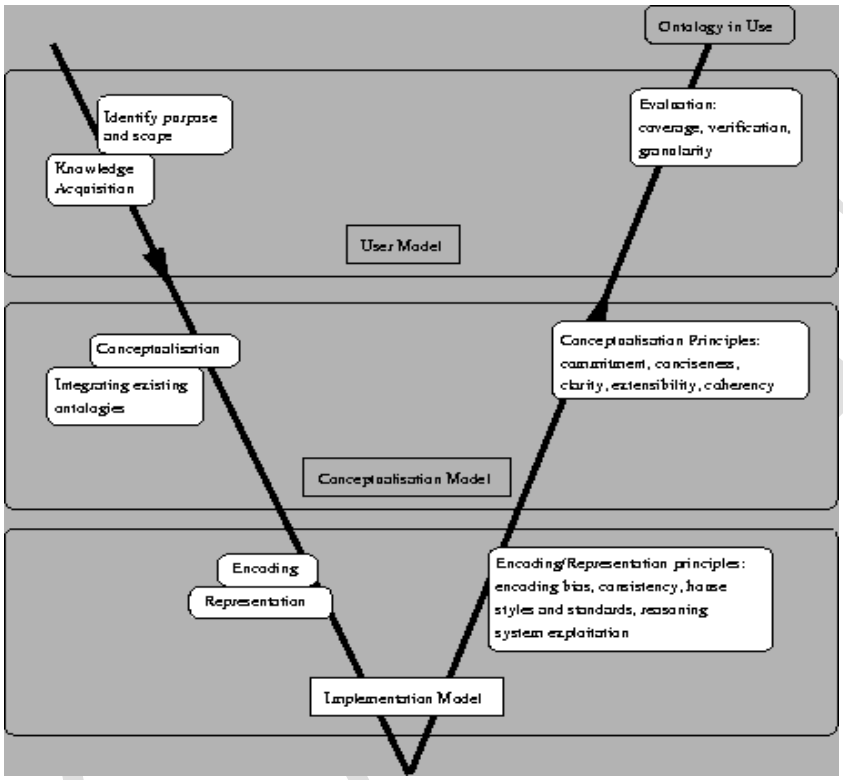


Figure 1: The V-model inspired methodology for building ontologies.

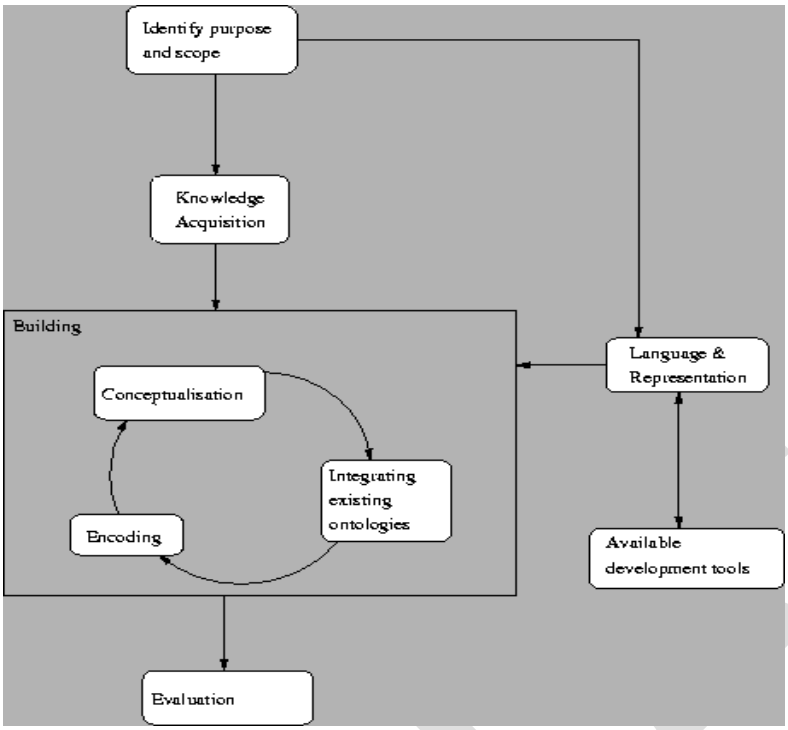


Figure 2: The ontology building life-cycle.

The stages in the V-process model and life-cycle are:

Identify purpose and scope:

developing a requirements specification for the ontology by identifying the intended scope and purpose of the ontology. A well-characterised requirements specification is important to the design, evaluation and re-use of an ontology. It can be seen from Section 4 that the use to which an ontology is put has a great effect on the content and style of that ontology.

Knowledge Acquisition:

the process of acquiring domain knowledge from which the ontology will be built. Sources span the complete range of knowledge holders: Specialist biologists; database metadata; standard text books; research papers and other ontologies. The EcoCyc and RiboWeb ontologies had the bulk of their knowledge gathered from the research literature on *E. coli.* metabolism and ribosomal structure respectively. In the former case this was a huge volume of material, which took many years to process. The TaO, being built to query databases, extracted a large part of its knowledge from database documentation. Standard texts also contributed to the knowledge of core molecular biology.

Conceptualisation:

identifying the key concepts that exist in the domain, their properties and the relationships that hold between them; identifying natural language terms to refer to such concepts, relations and attributes; and structuring domain knowledge into explicit conceptual models. The ontology is usually described using some informal terminology.

Integrating:

use or specialise an existing ontology: a task frequently hindered by the inadequate documentation of existing ontologies, notably their implicit assumptions. Using a generic ontology, such as MBO, gives a deeper definition of the concepts in the chosen domain.

Encoding:

representing the conceptualisation in some formal language, e.g. frames, object models or logic. This includes the creation of formal competency questions in terms of the terminological specification language chosen (usually first order logic). The representation of ontologies is explored further below.

Documentation:

informal and formal complete definitions, assumptions and examples are essential to promote the appropriate use and re-use of an ontology. Documentation is important for defining, more expansively than is possible within the ontology, the exact meaning of terms within the ontology.

Evaluation:

determining the appropriateness of an ontology for its intended application. Evaluation is done pragmatically, by assessing the competency of the ontology to satisfy the requirements of its application, including determining the consistency, completeness and conciseness of an ontology. Conciseness implies an absence of redundancy in the definitions of an ontology and an appropriate granularity. For example, an ontology that modelled protein molecules at the atomic resolution when the amino acid level would suffice would not be considered concise.

Ontology learning:

Ontology learning (ontology extraction, ontology generation, or ontology acquisition) is the automatic or semi-automatic creation of ontologies, including extracting the corresponding domain's terms and the relationships between the concepts that these terms represent from a corpus of natural language text, and encoding them with an ontology language for easy retrieval.

As building ontologies manually is extremely labor-intensive and time-consuming, there is great motivation to automate the process.

Typically, the process starts by extracting terms and concepts or noun phrases from plain text using linguistic processors such as part-of-speech tagging and phrase chunking. Then statistical[1] or symbolic [2][3] techniques are used to extract relation signatures, often based on pattern-based[4] or definition-based[5] hypernym extraction techniques.

UNIT:II

SYLLABUS

Ontological Engineering: Ontologies – Taxonomies –Topic Maps – Classifying Ontologies – Terminological aspects: concepts, terms, relations between them – Complex Objects –Subclasses and Sub-properties definitions – Upper Ontologies – Quality – Uses - Types of terminological resources for ontology building – Methods and methodologies for building ontologies – Multilingual Ontologies -Ontology Development process and Life cycle – Methods for Ontology Learning – Ontology Evolution – Versioning

Ontological Engineering:

Ontologies:

Ontology-explicit formal specifications of the terms in the domain and relations among them (Gruber 1993).

Ontology is a key technique with which to annotate semantics and provide a common, comprehensible foundation for resources on the Semantic.

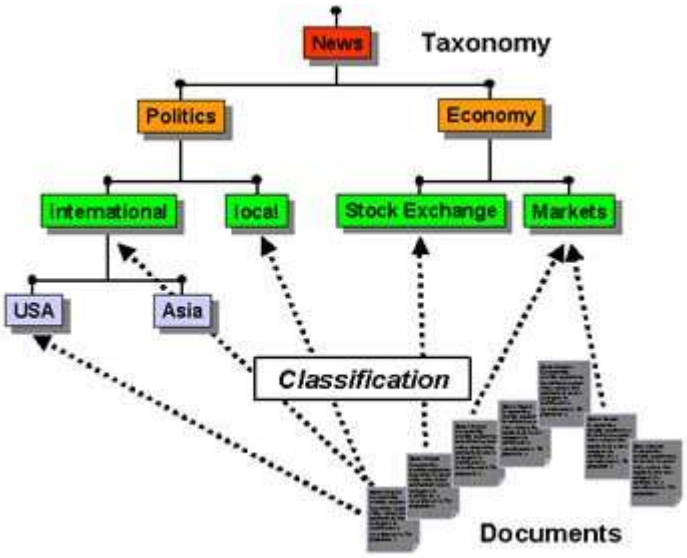
An ontology is a formal explicit description of concepts in a domain of discourse (classes (sometimes called concepts)), properties of each concept describing various features and attributes of the concept (slots (sometimes called roles or properties)), and restrictions on slots (facets (sometimes called role restrictions)). An ontology together with a set of individual instances of classes constitutes a knowledge base.

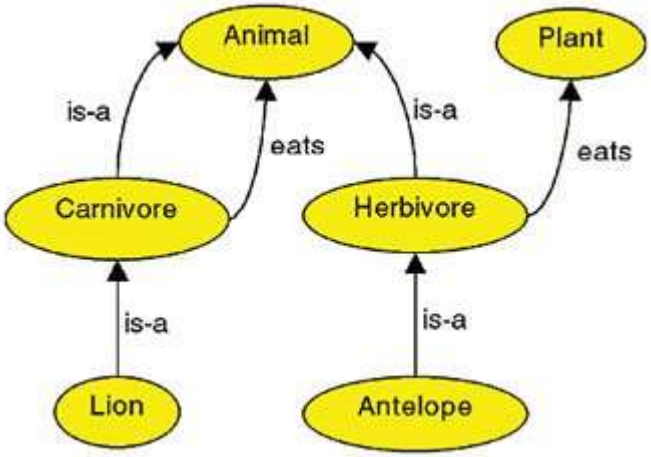
Taxonomy:

Taxonomy is usually only a hierarchy of concepts (i.e. the only relation between the concepts is parent/child, or subClass/superClass, or broader/narrower).

Taxonomy takes into consideration one type of relationship, whereas ontology takes into account many different complex relationships between the concepts.

The primary purpose of taxonomy and ontology is to help categorize and classify. Taxonomies are a system of classifying objects. Most taxonomies have a hierarchical structure, but it is not a requisite. It can be said that a taxonomy is simple hierarchical arrangement of entities. It utilizes a parent-child relation in its classification.





The main difference between Taxonomy and Ontology is that taxonomy is simpler in nature than ontology. Taxonomy takes into consideration one type of relationship, whereas ontology takes into account many different complex relationships between the concepts

Topic Maps:

Topic map is relatively a new entrant to this information space. Topic map standard describes how complex relationships between abstract concepts and real world resources can be represented using XML syntax.

A topic map is a standard for the representation and interchange of knowledge, with an emphasis on the findability of information. Topic maps were originally developed in the late 1990s as a way to represent back-of-the-book index structures so that multiple indexes from different sources could be merged. However, the developers quickly realized that with a little additional generalization, they could create a meta-model with potentially far wider application. The ISO standard is formally known as ISO/IEC 13250:2003.

A topic map represents information using

- topics, representing any concept, from people, countries, and organizations to software modules, individual files, and events,
- associations, representing hypergraph relationships between *topics*, and
- occurrences, representing information resources relevant to a particular *topic*.

Topic maps are similar to concept maps and mind maps in many respects, though only topic maps are ISO standards. Topic maps are a form of semantic web technology similar to RDF.

Terminological Aspects:

Terminology is the study of terms and their use. Terms are words and compound words or multi-word expressions that in specific contexts are given specific meanings—these may deviate from the meanings the same words have in other contexts and in everyday language. Terminology is a discipline that studies, among other things, the development of such terms and their interrelationships within a specialized domain. Terminology differs from lexicography, as it involves the study of concepts, conceptual systems and their labels (*terms*), whereas lexicography studies words and their meanings.

Terminology is a discipline that systematically studies the "labelling or designating of concepts" particular to one or more subject fields or domains of human activity. It does this through the research and analysis of terms in context for the purpose of documenting and promoting consistent usage. Terminology can be limited to one or more languages (for example,

"multilingual terminology" and "bilingual terminology"), or may have an interdisciplinarity focus on the use of terms in different fields.

Overview:

The discipline of terminology consists primarily of the following aspects:

- analyzing the concepts and concept structures used in a field or domain of activity
- identifying the terms assigned to the concepts
- in the case of bilingual or multilingual terminology, establishing correspondences between terms in the various languages
- compiling the terminology, on paper or in databases
- managing terminology databases
- creating new terms, as required

Types of terminology:

A distinction is made between two types of terminology work:

- Ad hoc work on terminology, which deals with a single term or a limited number of terms
- Systematic collection of terminology, which deals with all the terms in a specific subject field or domain of activity, often by creating a structured ontology of the terms within that domain and their interrelationships.

Ad hoc terminology is prevalent in the translation profession, where a translation for a specific term (or group of terms) is required quickly to solve a particular translation problem.

Ontology Development process and Life cycle:

Methodologies broadly divide into those that are stage-based (e.g. TOVE) and those that rely on iterative evolving prototypes (e.g. Methontology). These are in fact complementary techniques. Most distinguish between an informal stage, where the ontology is sketched out using either natural language descriptions or some diagram technique, and a formal stage where the ontology is encoded in a formal knowledge representation language, that is machine computable. As an ontology should ideally be communicated to people and unambiguously interpreted by software, the informal representation helps the former and the formal the latter.

Figures 1 and 2 represents a skeletal methodology and life-cycle for building ontologies, inspired by the software engineering V-process model . The left side of the V charts the processes in building an ontology and the right side charts the guidelines, principles and evaluation used to `quality assure' the ontology. The overall process, however, moves through a life-cycle, as depicted in Figure 2.

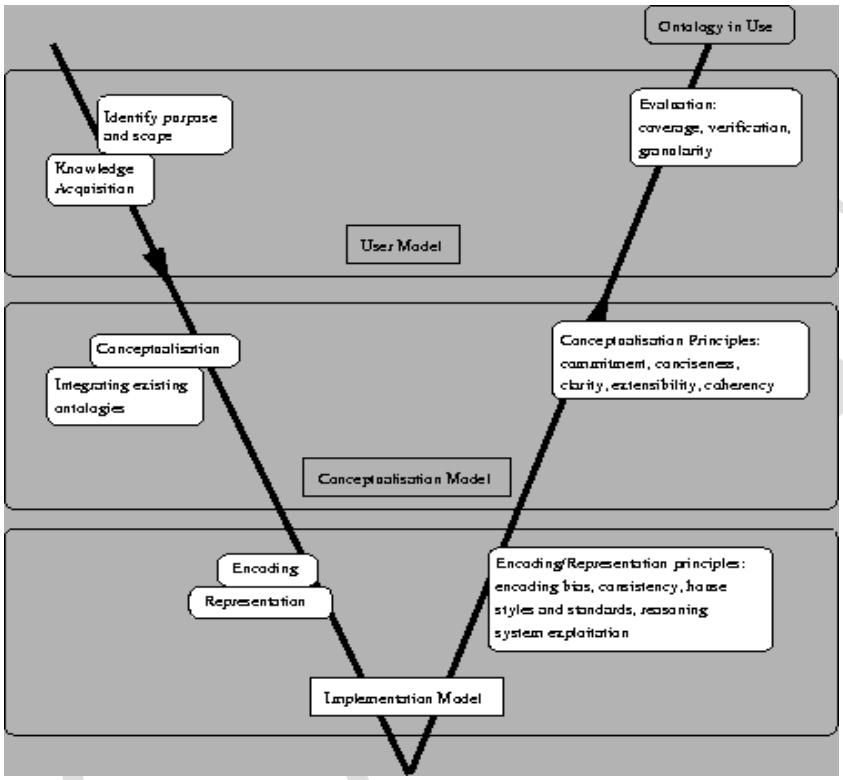


Figure 1: The V-model inspired methodology for building ontologies.

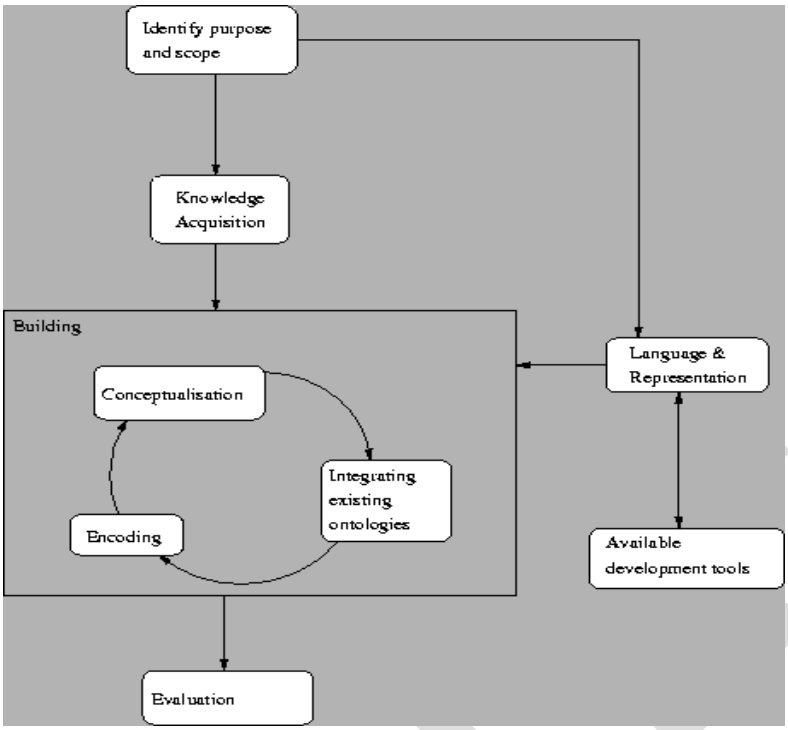


Figure 2: The ontology building life-cycle.

The stages in the V-process model and life-cycle are:

Identify purpose and scope:

developing a requirements specification for the ontology by identifying the intended scope and purpose of the ontology. A well-characterised requirements specification is important to the design, evaluation and re-use of an ontology. It can be seen from Section 4 that the use to which an ontology is put has a great effect on the content and style of that ontology.

Knowledge Acquisition:

the process of acquiring domain knowledge from which the ontology will be built. Sources span the complete range of knowledge holders: Specialist biologists; database metadata; standard text books; research papers and other ontologies. The EcoCyc and RiboWeb ontologies had the bulk of their knowledge gathered from the research literature on *E. coli.* metabolism and ribosomal structure respectively. In the former case this was a huge volume of material, which took many years to process. The TaO, being built to query databases, extracted a large part of its knowledge from database documentation. Standard texts also contributed to the knowledge of core molecular biology.

Conceptualisation:

identifying the key concepts that exist in the domain, their properties and the relationships that hold between them; identifying natural language terms to refer to such concepts, relations and attributes; and structuring domain knowledge into explicit conceptual models. The ontology is usually described using some informal terminology.

Integrating:

use or specialise an existing ontology: a task frequently hindered by the inadequate documentation of existing ontologies, notably their implicit assumptions. Using a generic ontology, such as MBO, gives a deeper definition of the concepts in the chosen domain.

Encoding:

representing the conceptualisation in some formal language, e.g. frames, object models or logic. This includes the creation of formal competency questions in terms of the terminological specification language chosen (usually first order logic). The representation of ontologies is explored further below.

Documentation:

informal and formal complete definitions, assumptions and examples are essential to promote the appropriate use and re-use of an ontology. Documentation is important for defining, more expansively than is possible within the ontology, the exact meaning of terms within the ontology.

Evaluation:

determining the appropriateness of an ontology for its intended application. Evaluation is done pragmatically, by assessing the competency of the ontology to satisfy the requirements of its application, including determining the consistency, completeness and conciseness of an ontology. Conciseness implies an absence of redundancy in the definitions of an ontology and an appropriate granularity. For example, an ontology that modelled protein molecules at the atomic resolution when the amino acid level would suffice would not be considered concise.

Ontology learning:

Ontology learning (ontology extraction, ontology generation, or ontology acquisition) is the automatic or semi-automatic creation of ontologies, including extracting the corresponding domain's terms and the relationships between the concepts that these terms represent from a corpus of natural language text, and encoding them with an ontology language for easy retrieval.

As building ontologies manually is extremely labor-intensive and time-consuming, there is great motivation to automate the process.

Typically, the process starts by extracting terms and concepts or noun phrases from plain text using linguistic processors such as part-of-speech tagging and phrase chunking. Then statistical[1] or symbolic [2][3] techniques are used to extract relation signatures, often based on pattern-based[4] or definition-based[5] hypernym extraction techniques.

SYLLABUS

Structuring And Describing Web Resources :Structured Web Documents - XML – Structuring – Namespaces – Addressing – Querying – Processing - RDF – RDF Data Model – Serialization Formats- RDF Vocabulary – Inferencing - RDFS – basic Idea – Classes – Properties- Utility Properties – RDFS Modeling for Combinations and Patterns- Transitivity

Ontology

Ontology is the study of the kinds of things that exist. In AI, the programs and sentences deal with various kinds of objects, and we study what these kinds are and what their basic properties are.

In general, ontology is the study or concern about what kinds of things exist - what entities there are in the universe. It is a branch of Meta physics, the study of first principles or the essence of things.

In **artificial intelligence (AI),** an ontology is, according to Tom Gruber, an AI specialist at Stanford University, **"the specification of conceptualizations, used to help programs and humans share knowledge."**

In the context of **computer and information sciences, ontology defines a set of representational primitives with which to model a domain of knowledge or discourse.**

The representational primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members).

In the context of **database systems**, ontology can be viewed as **a level of abstraction of data models,** analogous to hierarchical and relational models, but intended for modeling knowledge about individuals, their attributes, and their relationships to other individuals.

Uses:

Ontologies are used for integrating heterogeneous databases, enabling interoperability

among disparate systems, and specifying interfaces to independent, knowledge-based services. There are now standard languages and a variety of commercial and open source tools for creating and working with ontologies.

KEY APPLICATIONS

Ontologies are part of the W3C standards stack for the **Semantic Web**, in which they are used to **Specify standard conceptual vocabularies** in which to

- **exchange data among systems,**
- **provide services for answering queries,**
- **publish reusable knowledge bases,**
- **and offer services to facilitate interoperability across multiple heterogeneous systems and databases.**

FOAF

FOAF (an acronym of **Friend of a friend**) is a **machine-readable ontology** describing **persons, their activities and their relations to other people and objects**. Anyone can use FOAF to describe him or herself. FOAF allows groups of people to describe social networks without the need for a centralized database.

FOAF is a descriptive vocabulary expressed using the Resource Description Framework (RDF) and the Web Ontology Language (OWL). Computers may use these

FOAF profiles to find, for example, all people living in Europe, or to list all people both you and a friend of yours know. This is accomplished by defining relationships between people. Each profile has a unique identifier (such as the person's e-mail addresses, a Jabber ID, or a URI of the homepage or weblog of the person), which is used when defining these relationships. The FOAF project, which defines and extends the vocabulary of a FOAF profile, was started in 2000 by Libby Miller and Dan Brickley. It can be considered the first Social Semantic Web application, in that it combines RDF technology with 'Social Web' concerns.

Example:

The following FOAF profile (written in Turtle format) states that Jimmy Wales is the name of the person described here. His e-mail address, homepage and depiction are resources, which means that each can be described using RDF as well. He has Wikimedia as an interest, and knows Angela Beesley (which is the name of a 'Person' resource).

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
<#JW>
  a foaf:Person ;
  foaf:name "Jimmy Wales" ;
  foaf:mbox <mailto:jwales@bomis.com> ;
  foaf:homepage <http://www.jimmywales.com/> ;
  foaf:nick "Jimbo" ;
  foaf:depiction <http://www.jimmywales.com/aus_img_small.jpg> ;
  foaf:interest <http://www.wikimedia.org> ;
  foaf:knows [
    a foaf:Person ;
    foaf:name "Angela Beesley"
```

```
  ] .

<http://www.wikimedia.org>
  rdfs:label "Wikipedia" .
```

Turtle (syntax)

Turtle (Terse RDF Triple Language) is a **serialization format for Resource Description Framework (RDF) graphs.** A subset of Tim Berners-Lee and Dan Connolly's Notation3 (N3) language, it was defined by Dave Beckett, and is a superset of the minimal N-Triples format. Unlike full N3, Turtle doesn't go beyond RDF's graph model. SPARQL uses a similar N3 subset to Turtle for its graph patterns, but using N3's brace syntax for delimiting sub graphs. Turtle was accepted as a first working draft by the World Wide Web Consortium (W3C) RDF Working Group on 9 August 2011.

Turtle is popular among Semantic Web developers as a human-friendly alternative to RDF/XML. A significant proportion of RDF toolkits include Turtle parsing and serializing capability.

Some examples are Redland, Sesame, Jena and RDF Lib.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-
syntax-ns#> . @prefix dc:
<http://purl.org/dc/elements/1.1/> .
@prefix ex: <http://example.org/stuff/1.0/> .

<http://www.w3.org/TR/rdf-syntax-
  grammar> dc:title "RDF/XML Syntax
  Specification (Revised)" ; ex:editor [
```

Uniform Resource Identifier (URI)

In computing, a **uniform resource identifier** (**URI**) is a **string of characters used to identify a name or a resource. Such identification enables interaction with representations of the resource over a network (typically the World Wide Web) using specific protocols**.

Schemes specifying a concrete syntax and associated protocols define each URI.

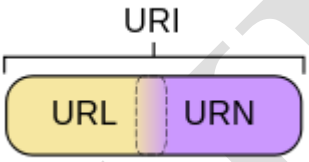


Diagram of URI scheme categories. Schemes in the **URL (locator) and URN (name)** categories form subsets of URI, and also (generally) disjoint sets. Technically URL and URN function as resource IDs; however, one cannot exactly categorize many schemes as one or the other: we can treat all URIs as names, and some schemes embody aspects of both categories.

Relationship to URL and URN

URIs can be classified as locators (URLs), as names (URNs), or as both. A uniform resource name(URN) functions like a person's name, while a uniform resource locator (URL) resembles that person's street address. In other words: the URN defines an item's identity, while the URL provides a method for finding it.

The ISBN system for uniquely identifying books provides a typical example of the use of URNs.ISBN 0-486-27557-4 (urn:isbn:0-486-27557-4) cites, unambiguously, a specific edition of Shakespeare's play *Romeo and Juliet*. To gain access to this object and read the book, one needs its location: a URL address. A typical URL for this book on a Unix-like operating system would be a file path such as file:///home/username/RomeoAndJuliet.pdf, identifying the electronic book saved in a file on a local hard disk. So URNs and URLs have complementary purposes.

Uses of URI references in markup languages

- In HTML, the value of the **src** attribute of the **img** element provides a URI reference, as does the value of the **href** attribute of **a** or **link** element.
- In XML, the system identifier appearing after the **SYSTEM** keyword in a **DTD** is a fragment less URI reference.
- In XSLT, the value of the href attribute of the xsl:import element/instruction is a URI reference; likewise the first argument to thedocument() function.

Examples of absolute URIs

- http://example.org/absolute/URI/with/absolute/path/to/resource.txt
- ftp://example.org/resource.txt
- urn:issn:1535-3613

Examples of URI references

- http://en.wikipedia.org/wiki/URI#Examples_of_URI_references ("http" specifies the 'scheme' name, "en.wikipedia.org" is the 'authority', "/wiki/URI" the 'path' pointing to this article, and "#Examples_of_URI_references" is a 'fragment' pointing to this section.)
- http://example.org/absolute/URI/with/absolute/path/to/resource.txt
- //example.org/scheme-relative/URI/with/absolute/path/to/resource.txt
- /relative/URI/with/absolute/path/to/resource.txt
- relative/path/to/resource.txt
- ../../../resource.txt
- ./resource.txt#frag01
- resource.txt
- #frag01
- *(empty string)*

Resource Description Framework (RDF)

The **Resource Description Framework** (**RDF**) is a family of World Wide Web Consortium (W3C) specifications originally designed as a metadata data model.

RDF is a standard model for data interchange on the Web. RDF has features **that facilitate data merging even if the underlying schemas differ**, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed.

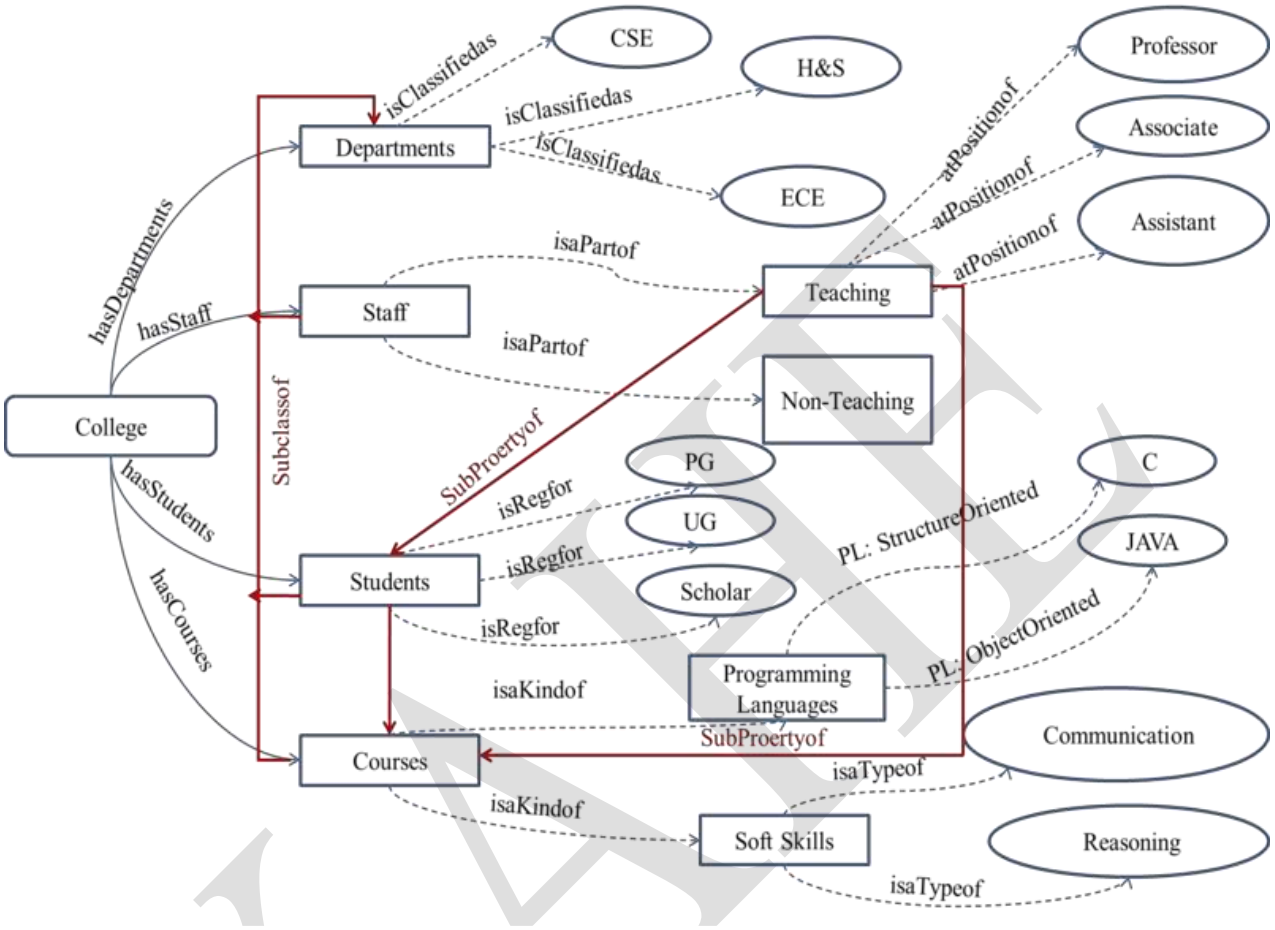
RDF extends the **linking structure of the Web** to use URIs to name the relationship between things as well as the two ends of the link (this is usually referred to as a "triple"). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications.

This linking structure forms a directed, labeled graph, where the edges represent the named link between two resources, represented by the graph nodes. This *graph view* is the easiest possible mental model for RDF and is often used in easy-to-understand visual explanations.

The RDF data model is similar to classic conceptual modeling approaches such as **entity-relationship or class diagrams**, as it is based upon the idea of making statements about resources (in particular Web resources) in the form of **subject-predicate-object expressions**. These expressions are known as ***triples* in RDF terminology**. The subject denotes the resource, and the predicate denotes traits or aspects of the resource and expresses a relationship between the subject and the object. For example, one way to represent the notion "The sky has the color blue" in RDF is as the triple: a subject denoting "the sky", a predicate denoting "has the color", and an object denoting "blue". Therefore RDF swaps object for subject that would be used in the classical notation of an Entity–attribute–value model within **Object oriented design**; object (sky), attribute (color) and value (blue). RDF is an abstract model with several serialization formats (i.e., file formats), and so the particular way in which a resource or triple is encoded varies from format to format.

A collection of RDF statements intrinsically represents a labeled, directed multi-graph. As such, an RDF-based data model is more naturally suited to certain kinds of knowledge representation than the relational model and other ontological models. However, in practice, RDF data is often persisted in relational database or native representations also called Triplestores, or Quad stores if context (i.e. the named graph) is also persisted for each RDF triple. As RDFS and OWL demonstrate, additional ontology languages can be built upon RDF.

Example:



RDF Vocabulary

The vocabulary defined by the RDF specification is as follows:

Classes

rdf

- **rdf:XMLLiteral** - the class of XML literal values
- **rdf:Property** - the class of properties
- **rdf:Statement** - the class of RDF statements
- **rdf:Alt**, **rdf:Bag**, **rdf:Seq** - containers of alternatives, unordered containers, and ordered containers (rdfs:Container is a super-class of the three)
- **rdf:List** - the class of RDF Lists

- **rdf:nil** - an instance of rdf:List representing the empty list

rdfs

- **rdfs:Resource** - the class resource, everything
- **rdfs:Literal** - the class of literal values, e.g. strings and integers
- **rdfs:Class** - the class of classes

- **rdfs:Datatype** - the class of RDF datatypes
- **rdfs:Container** - the class of RDF containers
- **rdfs:ContainerMembershipProperty** - the class of container membership properties, rdf:_1, rdf:_2, ..., all of which are sub-properties of rdfs:member

Properties

rdf

- **rdf:type** - an instance of rdf:Property used to state that a resource is an instance of a class
- **rdf:first** - the first item in the subject RDF list
- **rdf:rest** - the rest of the subject RDF list after rdf:first
- **rdf:value** - idiomatic property used for structured values
- **rdf:subject** - the subject of the subject RDF statement
- **rdf:predicate** - the predicate of the subject RDF statement
- **rdf:object** - the object of the subject RDF statement

rdf:Statement, rdf:subject, rdf:predicate, rdf:object are used for reification (see below).

rdfs

- **rdfs:subClassOf** - the subject is a subclass of a class
- **rdfs:subPropertyOf** - the subject is a subproperty of a property
- **rdfs:domain** - a domain of the subject property
- **rdfs:range** - a range of the subject property
- **rdfs:label** - a human-readable name for the subject
- **rdfs:comment** - a description of the subject resource

- **rdfs:member** - a member of the subject resource
- **rdfs:seeAlso** - further information about the subject resource
- **rdfs:isDefinedBy** - the definition of the subject resource

This vocabulary is used as a foundation for RDF Schema where it is extended.

Serialization formats

RDF/XML serialization	
	
Filename extension	.rdf
Internet media type	application/rdf+xml
Developed by	World Wide Web Consortium
Standard(s)	Concepts and Abstract Syntax February 10, 2004; 8 years ago
Open format?	Yes

Two common serialization formats are in use.

The first is an **XML format**. This format is often called simply RDF because it was introduced among the other W3C specifications defining RDF. However, it is important to distinguish the XML format from the abstract RDF model itself. Its MIME media type, application/rdf+xml, was registered by RFC 3870. It recommends RDF documents to follow the new 2004 specifications.

In addition to serializing RDF as XML, the W3C introduced **Notation 3 (or N3) as a non-XML serialization of RDF models** designed to be easier to write by hand, and in some cases easier to follow. Because it is based on a tabular notation, it makes the underlying triples encoded in the documents more easily recognizable compared to the XML serialization. N3 is closely related to the Turtle and N-Triples formats. Triples may be stored in a triplestore.

Resource identification

The subject of an RDF statement is either a Uniform Resource Identifier (URI) or a blank node, both of which denote resources. Resources indicated by blank nodes are called anonymous resources. They are not directly identifiable from the RDF statement. The

predicate is a URI which also indicates a resource, representing a relationship. The object is a URI, blank node or a Unicode string literal.

In Semantic Web applications, and in relatively popular applications of RDF like RSS and FOAF (Friend of a Friend), resources tend to be represented by URIs that intentionally denote, and can be used to access, actual data on the World Wide Web. But RDF, in general, is not limited to the description of Internet-based resources. In fact, the URI that names a resource does not have to be de reference able at all. For example, a URI that begins with "http:" and is used as the subject of an RDF statement does not necessarily have to represent a resource that is accessible via HTTP, nor does it need to represent a tangible, network-accessible resource — such a URI could represent absolutely anything. However, there is broad agreement that a bare URI (without a # symbol) which returns a 300-level coded response when used in an HTTP GET request should be treated as denoting the internet resource that it succeeds in accessing.

Therefore, producers and consumers of RDF statements must agree on the semantics of resource identifiers. Such agreement is not inherent to RDF itself, although there are some controlled vocabularies in common use, such as Dublin Core Metadata, which is partially mapped to a URI space for use in RDF. The intent of publishing RDF-based ontologies on the Web is often to establish, or circumscribe, the intended meanings of the resource identifiers used to express data in RDF. For example, the URI http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#Merlot is intended by its owners to refer to the class of all Merlot red wines by vintner (i.e., instances of http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#Merlot each represent the class of all wine produced by a single vintner), a definition which is expressed by the OWL ontology — itself an RDF document — in which it

occurs. Without careful analysis of the definition, one might erroneously conclude that an instance of http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#Merlot was something physical, instead of a type of wine.

Note that this is not a 'bare' resource identifier, but is rather a URI reference, containing the '#' character and ending with a fragment identifier.

RDF Design Goals

The design of RDF is intended to meet the following goals:

- having a simple data model
- having formal semantics and provable inference
- using an extensible URI-based vocabulary
- using an XML-based syntax
- supporting use of XML schema data types
- allowing anyone to make statements about any resource

Query and inference languages

The predominant query language for RDF graphs is SPARQL. SPARQL is an SQL-like language, and a recommendation of the W3C as of January 15, 2008.

An example of a SPARQL query to show country capitals in Africa, using a fictional ontology.

```
PREFIX abc: <nul://sparql/exampleOntology#> .
SELECT ?capital ?country
WHERE {
  ?x abc:cityname ?capital ;
     abc:isCapitalOf ?y.
  ?y abc:countryname ?country ;
     abc:isInContinent abc:Africa.
}
```

Other ways to query RDF graphs include:

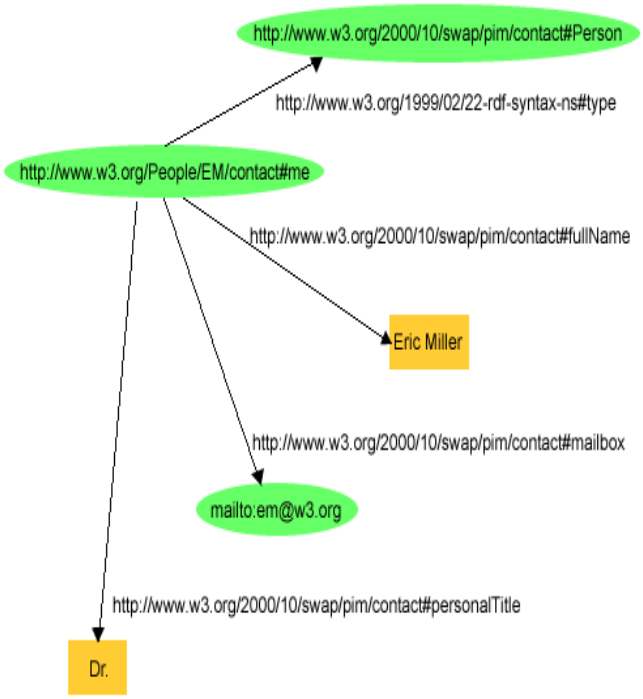
- RDQL, precursor to SPARQL, SQL-like
- Versa, compact syntax (non–SQL-like), solely implemented in 4Suite (Python)
- RQL, one of the first declarative languages for uniformly querying RDF schemas and resource descriptions, implemented in RDFSuite.
- SeRQL, part of Sesame
- XUL has a template element in which to declare rules for matching data in RDF. XUL uses RDF extensively for data binding.

##########

Examples

[edit]**Example 1: RDF Description of a person named Eric Miller**[17]

The following example is taken from the W3C website[17] describing a resource with statements "there is a Person identified byhttp://www.w3.org/People/EM/contact#me, whose name is Eric Miller, whose email address is em@w3.org, and whose title is Dr.".



An RDF Graph Describing Eric Miller

The resource "http://www.w3.org/People/EM/contact#me" is the subject.

The objects are:

- "Eric Miller" (with a predicate "whose name is"),
- em@w3.org (with a predicate "whose email address is"), and
- "Dr." (with a predicate "whose title is").

- The subject is a URI.

The predicates also have URIs. For example, the URI for each predicate:

- "whose name is" is http://www.w3.org/2000/10/swap/pim/contact#fullName,

- "whose email address is" is http://www.w3.org/2000/10/swap/pim/contact#mailbox,
- "whose title is" is http://www.w3.org/2000/10/swap/pim/contact#personalTitle.

In addition, the subject has a type (with URI http://www.w3.org/1999/02/22-rdf-syntax-ns#type), which is person (with URI http://www.w3.org/2000/10/swap/pim/contact#Person), and a mailbox (with URIhttp://www.w3.org/2000/10/swap/pim/contact#mailbox.)

Therefore, the following "subject, predicate, object" RDF triples can be expressed:

- http://www.w3.org/People/EM/contact#me, http://www.w3.org/2000/10/swap/pim/
- http://www.w3.org/People/EM/contact#me, http://www.w3.org/2000/10/swap/pim/contact#personalTitle, "Dr."
- http://www.w3.org/People/EM/contact#me, http://www.w3.org/1999/02/22-rdf-syntax-ns#type,http://www.w3.org/2000/10/swap/pim/contact#Person
- http://www.w3.org/People/EM/contact#me, http://www.w3.org/2000/10/swap/pim/contact#mailbox, em@w3.org

```
<urn:x-states:New%20York> <http://purl.org/dc/terms/alternative> "NY" .
```

In this example, "urn:x-states:New%20York" is the URI for a resource that denotes the U.S. state New York, "http://purl.org/dc/terms/alternative" is the URI for a predicate (whose human-readable definition can be found at here [18]), and "NY" is a literal string. Note that the URIs chosen here are not standard, and don't need to be, as long as their meaning is known to whatever is reading them.

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-
 rdf-syntax-ns#"
 xmlns:dcterms="http://purl.org/dc/terms/">
    <rdf:Description rdf:about="urn:x-states:New%20York">
            <dcterms:alternative>NY</dcterms:alternative>
```

Example 2: The postal abbreviation for New York

Certain concepts in RDF are taken from logic and linguistics, where subject-predicate and subject-predicate-object structures have meanings similar to, yet distinct from, the uses of those terms in RDF. This example demonstrates:

In the English language statement *'New York has the postal abbreviation NY'* , *'New York'* would be the subject, *'has the postal abbreviation'* the predicate and *'NY'* the object.

Encoded as an RDF triple, the subject and predicate would have to be resources named by URIs. The object could be a resource or literal element. For example, in the Notation 3 form of RDF, the statement might look like:

However, because of the restrictions on the syntax of QNames (such as dcterms:alternative above), there are some RDF graphs that are not representable with RDF/XML.

[edit]**Example 3: A Wikipedia article about Tony Benn**

To an English-speaking person, the same information could be represented simply as:
In a like manner, given that "http://en.wikipedia.org/wiki/Tony_Benn" identifies a particular
The title of this resource, which is published by Wikipedia, is 'Tony Benn'
resource (regardless of whether that URI could be traversed as a hyperlink, or whether the
However, RDF puts the information in a formal way that a machine can understand. The
resource is *actually* the Wikipedia article about Tony Benn), to say that the title of this
purpose of RDF is to provide an encoding and interpretation mechanism so
resource is "Tony Benn" and its publisher is "Wikipedia" would be two assertions that could
that resources can be described in a way that particular software can understand it; in other
be expressed as valid RDF statements. In the N-Triples form of RDF, these statements might
words, so that software can access and use information that it otherwise couldn't use.
look like the following:
Both versions of the statements above are wordy because one requirement for an RDF resource (as a subject or a predicate) is that it be unique. The subject resource must be

unique in an attempt to pinpoint the exact resource being described. The predicate needs to
<http://en.wikipedia.org/wiki/Tony_Benn> <http://purl.org/dc/elements/1.1/publisher>
be unique in order to reduce the chance that the idea of Title or Publisher will be ambiguous
"Wikipedia" .
to software working with the description. If the software
And these statements might be expressed in RDF/XML as:
recognizes *http://purl.org/dc/elements/1.1/title* (a specific definition for the concept of a title
<rdf:RDF
established by the Dublin CoreMetadata Initiative), it will also know that this title is different
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
from a land title or an honorary title or just the letters t-i-t-l-e put together.
xmlns:dc="http://purl.org/dc/elements/1.1/">

```
    <rdf:Description rdf:about="http://en.wikipedia.org/wiki/Tony_Benn">
        <dc:title>Tony Benn</dc:title>
        <dc:publisher>Wikipedia</dc:publisher>
    </rdf:Description>
</rdf:RDF>
```

The following example shows how such simple claims can be elaborated on, by combining multiple RDF vocabularies. Here, we note that the primary topic of the Wikipedia page is a "Person" whose name is "Tony Benn":

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:foaf="http://xmlns.com/foaf/0.1/"
 xmlns:dc="http://purl.org/dc/elements/1.1/">
    <rdf:Description rdf:about="http://en.wikipedia.org/wiki/Tony_Benn">
        <dc:title>Tony Benn</dc:title>
        <dc:publisher>Wikipedia</dc:publisher>
        <foaf:primaryTopic>
           <foaf:Person>
              <foaf:name>Tony Benn</foaf:name>
           </foaf:Person>
        </foaf:primaryTopic>
    </rdf:Description>
</rdf:RDF>
```

[edit]Applications

- Sigma - Application from DERI in National University of Ireland, Galway(NUIG).
- Creative Commons - Uses RDF to embed license information in web pages and mp3 files.
- DOAC (Description of a Career) - supplements FOAF to allow the sharing of résumé information.
- Enterprise Architect: MDG Technology for ODM (ODM supports RDF and OWL).
- FOAF (Friend of a Friend) - designed to describe people, their interests and interconnections.
- Haystack client - Semantic web browser from MIT CS & AI lab.[19]
- IDEAS Group - developing a formal 4D Ontology for Enterprise Architecture using RDF as the encoding.[20]
- Microsoft shipped a product, Connected Services Framework,[21] which provides RDF-based Profile Management capabilities.

- MusicBrainz - Publishes information about Music Albums.[22]
- NEPOMUK, an open-source software specification for a Social Semantic desktop uses RDF as a storage format for collected metadata. NEPOMUK is mostly known because of its integration into the KDE SC 4 desktop environment.

- RDF Site Summary - one of several "RSS" languages for publishing information about updates made to a web page; it is often used for disseminating news article summaries and sharing weblog content.
- ResumeRDF - developed to express information contained in a personal Resume or Curriculum Vitae (CV) on the Semantic Web. This includes information about work and academic experience, skills, etc.
- Simple Knowledge Organization System (SKOS) - a KR representation intended to support vocabulary/thesaurus applications
- SIOC (Semantically-Interlinked Online Communities) - designed to describe online communities and to create connections between Internet-based discussions from message boards, weblogs and mailing lists.[23]
- Smart-M3 - provides an infrastructure for using RDF and specifically uses the ontology agnostic nature of RDF to enable heterogeneous mashing-up of information[24]
- Many other RDF schemas are available by searching SchemaWeb.[25]

Some uses of RDF include research into social networking. This is important because it could help governments keep track of undesirables. It will also help people in business fields understand better their relationships with members of industries that could be of use for product placement.[26] It will also help scientists understand how people are connected to one another.

RDF is being used to have a better understanding of traffic patterns. This is because the information regarding traffic patterns is on different websites, and RDF is used to integrate information from different sources on the web. Before, the common methodology was using keyword searching, but this method is problematic because it does not consider synonyms. This is why ontologies are useful in this situation. But one of the issues that comes up when trying to efficiently study traffic is that to fully understand traffic, concepts related to people, streets, and roads must be well understood. Since these are human concepts, they require the addition of fuzzy logic. This is because values that are useful when describing roads, like slipperiness, are not precise concepts and cannot be measured. This would imply that the best solution would incorporate both fuzzy logic and ontology.[27]

RDF Schema

RDF Schema (variously abbreviated as **RDFS**, **RDF(S)**, **RDF-S**, or **RDF/S**) is a set of classes with certain properties using the RDF extensible representation language, providing basic elements for the description of ontologies, otherwise called RDF vocabularies, intended to

structure RDF resources. These resources can be saved in a triple store to reach them with the query language SPARQL.

Main RDFS constructs

RDFS constructs are the RDFS classes, associated properties and utility properties built on the limited vocabulary of RDF.

[edit]**Classes**

- **rdfs:Resource** is the class of everything. All things described by RDF are resources.
- **rdfs:Class** declares a resource as a class for other resources.

A typical example of an rdfs:Class is foaf:Person in the Friend of a Friend (FOAF) vocabulary. An instance of foaf:Person is a resource that is linked to the class foaf:Person using the rdf:type property, such as in the following formal expression of the natural language sentence : 'John is a Person'.

ex:John rdf:type foaf:Person

The definition of rdfs:Class is recursive: rdfs:Class is the rdfs:Class of any rdfs:Class.

The other classes described by the RDF and RDFS specifications are:

- **rdfs:Literal** – literal values such as strings and integers. Property values such as textual strings are examples of RDF literals. Literals may be plain or typed.
- **rdfs:Datatype** – the class of datatypes. rdfs:Datatype is both an instance of and a subclass of rdfs:Class. Each instance of rdfs:Datatype is a subclass of rdfs:Literal.
- **rdf:XMLLiteral** – the class of XML literal values. rdf:XMLLiteral is an instance of rdfs:Datatype (and thus a subclass of rdfs:Literal).
- **rdf:Property** – the class of properties.

[edit]**Properties**

Properties are instances of the class rdf:Property and describe a relation between subject resources and object resources. When used as such a property is a predicate (see also RDF: reification).

- **rdfs:domain** of an rdf:predicate declares the class of the *subject* in a triple whose second component is the *predicate*.
- **rdfs:range** of an rdf:predicate declares the class or datatype of the *object* in a triple whose second component is the predicate.

For example, the following declarations are used to express that the property ex:employer relates a subject, which is of typefoaf:Person, to an object, which is of type foaf:Organization:

ex:employer rdfs:domain foaf:Person

ex:employer rdfs:range foaf:Organization

Given the previous two declarations, the following triple requires that ex:John is necessarily a foaf:Person, and ex:CompanyX is necessarily a foaf:Organization:

ex:John ex:employer ex:CompanyX

- **rdf:type** is a property used to state that a resource is an instance of a class.
- **rdfs:subClassOf** allows to declare hierarchies of classes.

For example, the following declares that 'Every Person is an Agent':

foaf:Person rdfs:subClassOf foaf:Agent

Hierarchies of classes support inheritance of a property domain and range (see definitions in next section) from a class to its subclasses.

- **rdfs:subPropertyOf** is an instance of rdf:Property that is used to state that all resources related by one property are also related by another.
- **rdfs:label** is an instance of rdf:Property that may be used to provide a human-readable version of a resource's name.
- **rdfs:comment** is an instance of rdf:Property that may be used to provide a human-readable description of a resource.

[edit]**Utility Properties**

- **rdfs:seeAlso** is an instance of rdf:Property that is used to indicate a resource that might provide additional information about the subject resource.

- **rdfs:isDefinedBy** is an instance of rdf:Property that is used to indicate a resource defining the subject resource. This property may be used to indicate an RDF vocabulary in which a resource is described.

[edit]RDFS Entailment

An entailment regime defines by RDFs (,OWL, etc.) not only which entailment relation is used, but also which queries and graphs are well-formed for the regime. The RDFS entailment is a

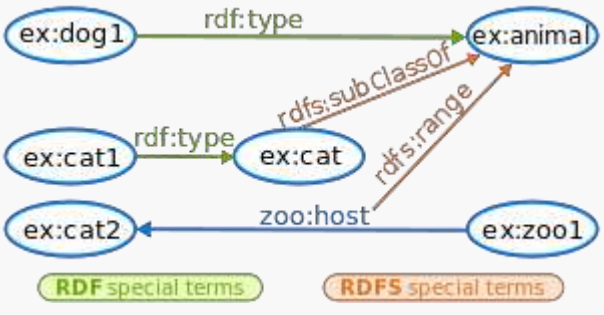
```
ex:cat      rdfs:subClassOf      ex:animal
```

Voila, the correct example:

For example, the following declares that 'Dog is an animal', 'Cat1 is a cat', 'Zoos host animals' and 'Zoo1 hosts the Cat2' :

```
ex:dog1      rdf:type          ex:animal
ex:cat1      rdf:type          ex:cat
zoo:host     rdfs:range        ex:animal
ex:zoo1      zoo:host          ex:cat2
```

But this graph is not well formed because the system can not guess that a cat is an animal. We have to add 'Cats are animals' to do a well-formed graph with :

In english	The graph
▪ Dog1 is an animal ▪ Cat1 is a cat ▪ Cats are animals ▪ Zoos host animals ▪ Zoo1 hosts the Cat2	

RDF/turtle

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex: <http://example.org/> .
@prefix zoo: <http://example.org/zoo/> .
ex:dog1    rdf:type        ex:animal .
ex:cat1    rdf:type        ex:cat .

ex:cat     rdfs:subClassOf ex:animal .
zoo:host   rdfs:range      ex:animal .
ex:zoo1    zoo:host        ex:cat2 .
```

If your triplestore (or RDF database) implements the regime entailment of RDF and RDFS, the SPARQL query as follows (the keyword "a" is equivalent to rdf:type in SPARQL):

```
PREFIX ex: <http://example.org/>
SELECT ?animal
```

```
WHERE
 { ?animal a ex:animal . }
```

Give the following result with *cat2* because the Cat's type inherits of Animal's type:

animal
<http://example.org/dog1>
<http://example.org/cat1>
<http://example.org/cat2>

RDFS Example

The following example demonstrates some of the RDFS facilities:

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xml:base="http://www.animals.fake/animals#">

<rdf:Description rdf:ID="animal">
        <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>

<rdf:Description rdf:ID="horse">
        <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
                <rdfs:subClassOf rdf:resource="#animal"/>
```

```
</rdf:Description>

</rdf:RDF>
```

Example Abbreviated

Since an RDFS class is an RDF resource we can abbreviate the example above by using rdfs:Class instead of rdf:Description, and drop the rdf:type information:

```
<?xml version="1.0"?>

<rdf:RDF

xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xml:base="http://www.animals.fake/animals#">

<rdfs:Class rdf:ID="animal" />

<rdfs:Class rdf:ID="horse">
    <rdfs:subClassOf rdf:resource="#animal"/>
</rdfs:Class>

</rdf:RDF>
```

XML Schema

An XML Schema describes the structure of an XML document.

XML Schema Example

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

XML Schema is an XML-based alternative to DTD.

An XML schema describes the structure of an XML document.

The XML Schema language is also referred to as XML Schema Definition (XSD).

What is an XML Schema?

The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD.

An XML Schema:

- defines elements that can appear in a document
- defines attributes that can appear in a document
- defines which elements are child elements
- defines the order of child elements
- defines the number of child elements

- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes

XML Schemas are the Successors of DTDs

We think that very soon XML Schemas will be used in most Web applications as a replacement for DTDs. Here are some reasons:

- XML Schemas are extensible to future additions
- XML Schemas are richer and more powerful than DTDs
- XML Schemas are written in XML
- XML Schemas support data types
- XML Schemas support namespaces

XML Schemas are much more powerful than DTDs.

XML Schemas Support Data Types

One of the greatest strength of XML Schemas is the support for data types.
With support for data types:

- It is easier to describe allowable document content
- It is easier to validate the correctness of data
- It is easier to work with data from a database
- It is easier to define data facets (restrictions on data)
- It is easier to define data patterns (data formats)
- It is easier to convert data between different data types

XML Schemas use XML Syntax

Another great strength about XML Schemas is that they are written in XML.

Some benefits of that XML Schemas are written in XML:

- You don't have to learn a new language
- You can use your XML editor to edit your Schema files
- You can use your XML parser to parse your Schema files
- You can manipulate your Schema with the XML DOM
- You can transform your Schema with XSLT

XML Schemas Secure Data Communication

When sending data from a sender to a receiver, it is essential that both parts have the same "expectations" about the content.

With XML Schemas, the sender can describe the data in a way that the receiver will understand.

A date like: "03-11-2004" will, in some countries, be interpreted as 3.November and in other countries as 11.March.

However, an XML element with a data type like this:

<date type="date">2004-03-11</date>

ensures a mutual understanding of the content, because the XML data type "date" requires the format "YYYY-MM-DD".

XML Schemas are Extensible

XML Schemas are extensible, because they are written in XML.
With an extensible Schema definition you can:

- Reuse your Schema in other Schemas
- Create your own data types derived from the standard types
- Reference multiple schemas in the same document

Well-Formed is not enough

A well-formed XML document is a document that conforms to the XML syntax rules, like:

- it must begin with the XML declaration
- it must have one unique root element
- start-tags must have matching end-tags
- elements are case sensitive
- all elements must be closed
- all elements must be properly nested
- all attribute values must be quoted
- entities must be used for special characters

Even if documents are well-formed they can still contain errors, and those errors can have serious consequences.

XML documents can have a reference to a DTD or to an XML Schema.

A Simple XML Document

Look at this simple XML document called "note.xml":

```
<?xml version="1.0"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

A DTD File

The following example is a DTD file called "note.dtd" that defines the elements of the XML document above ("note.xml"):

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

The first line defines the note element to have four child elements: "to, from, heading, body".
Line 2-5 defines the to, from, heading, body elements to be of type "#PCDATA".

An XML Schema

The following example is an XML Schema file called "note.xsd" that defines the elements of the XML document above ("note.xml"):

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">


<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>


</xs:schema>
```

The note element is a **complex type** because it contains other elements. The other elements (to, from, heading, body) are **simple types** because they do not contain other elements. You will learn more about simple and complex types in the following chapters.

A Reference to a DTD

This XML document has a reference to a DTD:

```
<?xml version="1.0"?>

<!DOCTYPE note SYSTEM
"http://www.w3schools.com/dtd/note.dtd">

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

A Reference to an XML Schema

This XML document has a reference to an XML Schema:

```
<?xml version="1.0"?>

<note
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

SYLLABUS

Structuring And Describing Web Resources :Structured Web Documents - XML – Structuring – Namespaces – Addressing – Querying – Processing - RDF – RDF Data Model – Serialization Formats- RDF Vocabulary – Inferencing - RDFS – basic Idea – Classes – Properties- Utility Properties – RDFS Modeling for Combinations and Patterns- Transitivity

Ontology

Ontology is the study of the kinds of things that exist. In AI, the programs and sentences deal with various kinds of objects, and we study what these kinds are and what their basic properties are.

In general, ontology is the study or concern about what kinds of things exist - what entities there are in the universe. It is a branch of Meta physics, the study of first principles or the essence of things.

In **artificial intelligence (AI),** an ontology is, according to Tom Gruber, an AI specialist at Stanford University, **"the specification of conceptualizations, used to help programs and humans share knowledge."**

In the context of **computer and information sciences, ontology defines a set of representational primitives with which to model a domain of knowledge or discourse.**

The representational primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members).

In the context of **database systems**, ontology can be viewed as **a level of abstraction of data models,** analogous to hierarchical and relational models, but intended for modeling knowledge about individuals, their attributes, and their relationships to other individuals.

Uses:

Ontologies are used for integrating heterogeneous databases, enabling interoperability

among disparate systems, and specifying interfaces to independent, knowledge-based services. There are now standard languages and a variety of commercial and open source tools for creating and working with ontologies.

KEY APPLICATIONS

Ontologies are part of the W3C standards stack for the **Semantic Web**, in which they are used to **Specify standard conceptual vocabularies** in which to

- **exchange data among systems,**
- **provide services for answering queries,**
- **publish reusable knowledge bases,**
- **and offer services to facilitate interoperability across multiple heterogeneous systems and databases.**

FOAF

FOAF (an acronym of **Friend of a friend**) is a **machine-readable ontology** describing **persons, their activities and their relations to other people and objects**. Anyone can use FOAF to describe him or herself. FOAF allows groups of people to describe social networks without the need for a centralized database.

FOAF is a descriptive vocabulary expressed using the Resource Description Framework (RDF) and the Web Ontology Language (OWL). Computers may use these

FOAF profiles to find, for example, all people living in Europe, or to list all people both you and a friend of yours know. This is accomplished by defining relationships between people. Each profile has a unique identifier (such as the person's e-mail addresses, a Jabber ID, or a URI of the homepage or weblog of the person), which is used when defining these relationships. The FOAF project, which defines and extends the vocabulary of a FOAF profile, was started in 2000 by Libby Miller and Dan Brickley. It can be considered the first Social Semantic Web application, in that it combines RDF technology with 'Social Web' concerns.

Example:

The following FOAF profile (written in Turtle format) states that Jimmy Wales is the name of the person described here. His e-mail address, homepage and depiction are resources, which means that each can be described using RDF as well. He has Wikimedia as an interest, and knows Angela Beesley (which is the name of a 'Person' resource).

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
<#JW>
  a foaf:Person ;
  foaf:name "Jimmy Wales" ;
  foaf:mbox <mailto:jwales@bomis.com> ;
  foaf:homepage <http://www.jimmywales.com/> ;
  foaf:nick "Jimbo" ;
  foaf:depiction <http://www.jimmywales.com/aus_img_small.jpg> ;
  foaf:interest <http://www.wikimedia.org> ;
  foaf:knows [
    a foaf:Person ;
    foaf:name "Angela Beesley"
```

```
  ] .

<http://www.wikimedia.org>
  rdfs:label "Wikipedia" .
```

Turtle (syntax)

Turtle (Terse RDF Triple Language) is a **serialization format for Resource Description Framework (RDF) graphs.** A subset of Tim Berners-Lee and Dan Connolly's Notation3 (N3) language, it was defined by Dave Beckett, and is a superset of the minimal N-Triples format. Unlike full N3, Turtle doesn't go beyond RDF's graph model. SPARQL uses a similar N3 subset to Turtle for its graph patterns, but using N3's brace syntax for delimiting sub graphs. Turtle was accepted as a first working draft by the World Wide Web Consortium (W3C) RDF Working Group on 9 August 2011.

Turtle is popular among Semantic Web developers as a human-friendly alternative to RDF/XML. A significant proportion of RDF toolkits include Turtle parsing and serializing capability.

Some examples are Redland, Sesame, Jena and RDF Lib.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-
syntax-ns#> . @prefix dc:
<http://purl.org/dc/elements/1.1/> .
@prefix ex: <http://example.org/stuff/1.0/> .

<http://www.w3.org/TR/rdf-syntax-
  grammar> dc:title "RDF/XML Syntax
  Specification (Revised)" ; ex:editor [
```

Uniform Resource Identifier (URI)

In computing, a **uniform resource identifier** (**URI**) is a **string of characters used to identify a name or a resource. Such identification enables interaction with representations of the resource over a network (typically the World Wide Web) using specific protocols**.

Schemes specifying a concrete syntax and associated protocols define each URI.

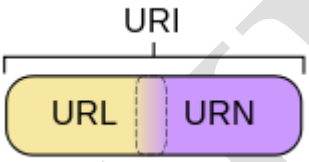


Diagram of URI scheme categories. Schemes in the **URL (locator) and URN (name)** categories form subsets of URI, and also (generally) disjoint sets. Technically URL and URN function as resource IDs; however, one cannot exactly categorize many schemes as one or the other: we can treat all URIs as names, and some schemes embody aspects of both categories.

Relationship to URL and URN

URIs can be classified as locators (URLs), as names (URNs), or as both. A uniform resource name(URN) functions like a person's name, while a uniform resource locator (URL) resembles that person's street address. In other words: the URN defines an item's identity, while the URL provides a method for finding it.

The ISBN system for uniquely identifying books provides a typical example of the use of URNs.ISBN 0-486-27557-4 (urn:isbn:0-486-27557-4) cites, unambiguously, a specific edition of Shakespeare's play *Romeo and Juliet*. To gain access to this object and read the book, one needs its location: a URL address. A typical URL for this book on a Unix-like operating system would be a file path such as file:///home/username/RomeoAndJuliet.pdf, identifying the electronic book saved in a file on a local hard disk. So URNs and URLs have complementary purposes.

Uses of URI references in markup languages

- In HTML, the value of the **src** attribute of the **img** element provides a URI reference, as does the value of the **href** attribute of **a** or **link** element.
- In XML, the system identifier appearing after the **SYSTEM** keyword in a **DTD** is a fragment less URI reference.
- In XSLT, the value of the href attribute of the xsl:import element/instruction is a URI reference; likewise the first argument to thedocument() function.

Examples of absolute URIs

- http://example.org/absolute/URI/with/absolute/path/to/resource.txt
- ftp://example.org/resource.txt
- urn:issn:1535-3613

Examples of URI references

- http://en.wikipedia.org/wiki/URI#Examples_of_URI_references ("http" specifies the 'scheme' name, "en.wikipedia.org" is the 'authority', "/wiki/URI" the 'path' pointing to this article, and "#Examples_of_URI_references" is a 'fragment' pointing to this section.)
- http://example.org/absolute/URI/with/absolute/path/to/resource.txt
- //example.org/scheme-relative/URI/with/absolute/path/to/resource.txt
- /relative/URI/with/absolute/path/to/resource.txt
- relative/path/to/resource.txt
- ../../../resource.txt
- ./resource.txt#frag01
- resource.txt
- #frag01
- *(empty string)*

Resource Description Framework (RDF)

The **Resource Description Framework** (**RDF**) is a family of World Wide Web Consortium (W3C) specifications originally designed as a metadata data model.

RDF is a standard model for data interchange on the Web. RDF has features **that facilitate data merging even if the underlying schemas differ**, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed.

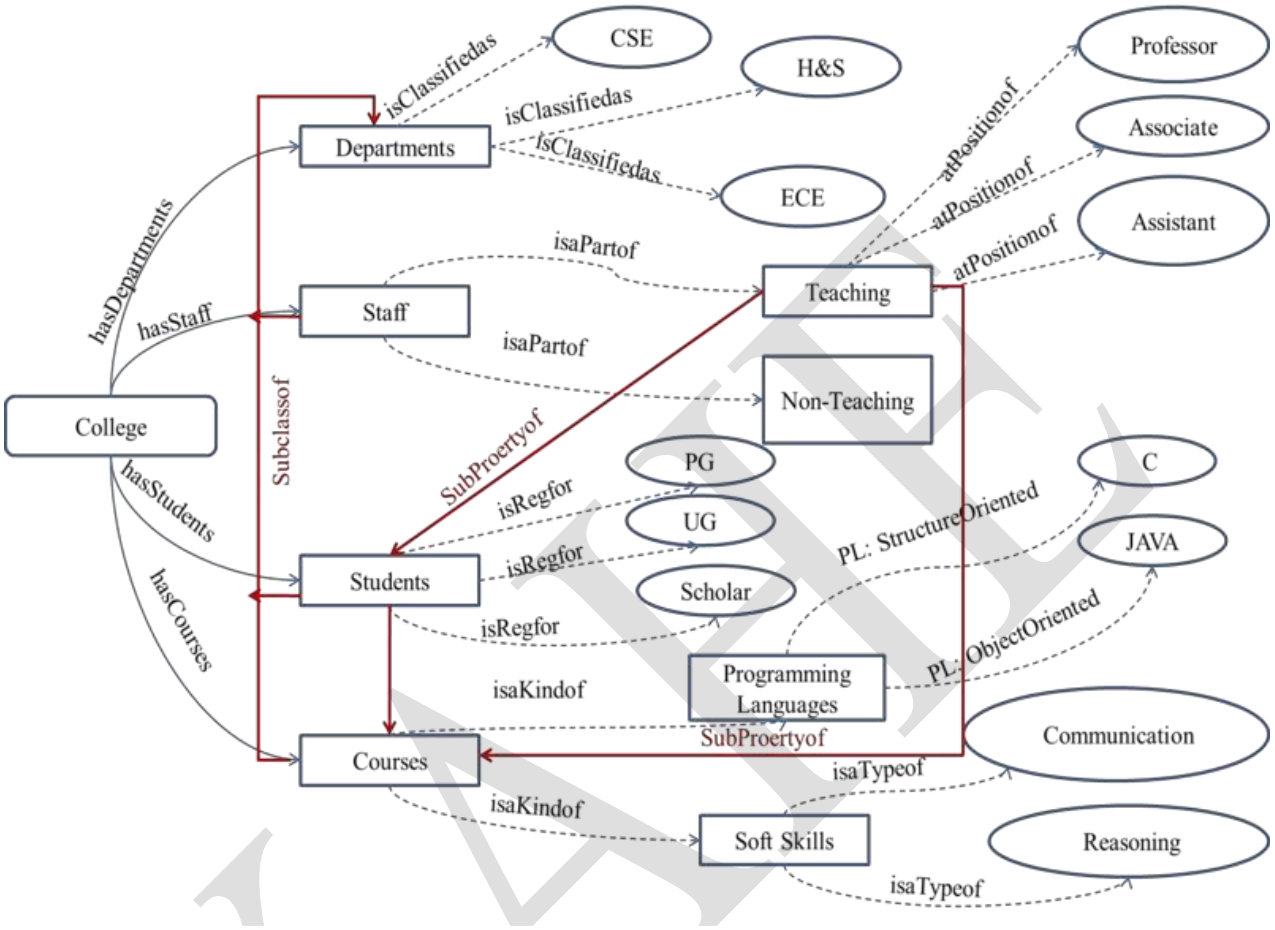
RDF extends the **linking structure of the Web** to use URIs to name the relationship between things as well as the two ends of the link (this is usually referred to as a "triple"). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications.

This linking structure forms a directed, labeled graph, where the edges represent the named link between two resources, represented by the graph nodes. This *graph view* is the easiest possible mental model for RDF and is often used in easy-to-understand visual explanations.

The RDF data model is similar to classic conceptual modeling approaches such as **entity-relationship or class diagrams**, as it is based upon the idea of making statements about resources (in particular Web resources) in the form of **subject-predicate-object expressions**. These expressions are known as ***triples* in RDF terminology**. The subject denotes the resource, and the predicate denotes traits or aspects of the resource and expresses a relationship between the subject and the object. For example, one way to represent the notion "The sky has the color blue" in RDF is as the triple: a subject denoting "the sky", a predicate denoting "has the color", and an object denoting "blue". Therefore RDF swaps object for subject that would be used in the classical notation of an Entity–attribute–value model within **Object oriented design**; object (sky), attribute (color) and value (blue). RDF is an abstract model with several serialization formats (i.e., file formats), and so the particular way in which a resource or triple is encoded varies from format to format.

A collection of RDF statements intrinsically represents a labeled, directed multi-graph. As such, an RDF-based data model is more naturally suited to certain kinds of knowledge representation than the relational model and other ontological models. However, in practice, RDF data is often persisted in relational database or native representations also called Triplestores, or Quad stores if context (i.e. the named graph) is also persisted for each RDF triple. As RDFS and OWL demonstrate, additional ontology languages can be built upon RDF.

Example:



RDF Vocabulary

The vocabulary defined by the RDF specification is as follows:

Classes

rdf

- **rdf:XMLLiteral** - the class of XML literal values
- **rdf:Property** - the class of properties
- **rdf:Statement** - the class of RDF statements
- **rdf:Alt**, **rdf:Bag**, **rdf:Seq** - containers of alternatives, unordered containers, and ordered containers (rdfs:Container is a super-class of the three)
- **rdf:List** - the class of RDF Lists

- **rdf:nil** - an instance of rdf:List representing the empty list

rdfs

- **rdfs:Resource** - the class resource, everything
- **rdfs:Literal** - the class of literal values, e.g. strings and integers
- **rdfs:Class** - the class of classes

- **rdfs:Datatype** - the class of RDF datatypes
- **rdfs:Container** - the class of RDF containers
- **rdfs:ContainerMembershipProperty** - the class of container membership properties, rdf:_1, rdf:_2, ..., all of which are sub-properties of rdfs:member

Properties

rdf

- **rdf:type** - an instance of rdf:Property used to state that a resource is an instance of a class
- **rdf:first** - the first item in the subject RDF list
- **rdf:rest** - the rest of the subject RDF list after rdf:first
- **rdf:value** - idiomatic property used for structured values
- **rdf:subject** - the subject of the subject RDF statement
- **rdf:predicate** - the predicate of the subject RDF statement
- **rdf:object** - the object of the subject RDF statement

rdf:Statement, rdf:subject, rdf:predicate, rdf:object are used for reification (see below).

rdfs

- **rdfs:subClassOf** - the subject is a subclass of a class
- **rdfs:subPropertyOf** - the subject is a subproperty of a property
- **rdfs:domain** - a domain of the subject property
- **rdfs:range** - a range of the subject property
- **rdfs:label** - a human-readable name for the subject
- **rdfs:comment** - a description of the subject resource

- **rdfs:member** - a member of the subject resource
- **rdfs:seeAlso** - further information about the subject resource
- **rdfs:isDefinedBy** - the definition of the subject resource

This vocabulary is used as a foundation for RDF Schema where it is extended.

Serialization formats

RDF/XML serialization	
	
Filename extension	.rdf
Internet media type	application/rdf+xml
Developed by	World Wide Web Consortium
Standard(s)	Concepts and Abstract Syntax February 10, 2004; 8 years ago
Open format?	Yes

Two common serialization formats are in use.

The first is an **XML format**. This format is often called simply RDF because it was introduced among the other W3C specifications defining RDF. However, it is important to distinguish the XML format from the abstract RDF model itself. Its MIME media type, application/rdf+xml, was registered by RFC 3870. It recommends RDF documents to follow the new 2004 specifications.

In addition to serializing RDF as XML, the W3C introduced **Notation 3 (or N3) as a non-XML serialization of RDF models** designed to be easier to write by hand, and in some cases easier to follow. Because it is based on a tabular notation, it makes the underlying triples encoded in the documents more easily recognizable compared to the XML serialization. N3 is closely related to the Turtle and N-Triples formats. Triples may be stored in a triplestore.

Resource identification

The subject of an RDF statement is either a Uniform Resource Identifier (URI) or a blank node, both of which denote resources. Resources indicated by blank nodes are called anonymous resources. They are not directly identifiable from the RDF statement. The

predicate is a URI which also indicates a resource, representing a relationship. The object is a URI, blank node or a Unicode string literal.

In Semantic Web applications, and in relatively popular applications of RDF like RSS and FOAF (Friend of a Friend), resources tend to be represented by URIs that intentionally denote, and can be used to access, actual data on the World Wide Web. But RDF, in general, is not limited to the description of Internet-based resources. In fact, the URI that names a resource does not have to be de reference able at all. For example, a URI that begins with "http:" and is used as the subject of an RDF statement does not necessarily have to represent a resource that is accessible via HTTP, nor does it need to represent a tangible, network-accessible resource — such a URI could represent absolutely anything. However, there is broad agreement that a bare URI (without a # symbol) which returns a 300-level coded response when used in an HTTP GET request should be treated as denoting the internet resource that it succeeds in accessing.

Therefore, producers and consumers of RDF statements must agree on the semantics of resource identifiers. Such agreement is not inherent to RDF itself, although there are some controlled vocabularies in common use, such as Dublin Core Metadata, which is partially mapped to a URI space for use in RDF. The intent of publishing RDF-based ontologies on the Web is often to establish, or circumscribe, the intended meanings of the resource identifiers used to express data in RDF. For example, the URI http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#Merlot is intended by its owners to refer to the class of all Merlot red wines by vintner (i.e., instances of http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#Merlot each represent the class of all wine produced by a single vintner), a definition which is expressed by the OWL ontology — itself an RDF document — in which it

occurs. Without careful analysis of the definition, one might erroneously conclude that an instance of http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#Merlot was something physical, instead of a type of wine.

Note that this is not a 'bare' resource identifier, but is rather a URI reference, containing the '#' character and ending with a fragment identifier.

RDF Design Goals

The design of RDF is intended to meet the following goals:

- having a simple data model
- having formal semantics and provable inference
- using an extensible URI-based vocabulary
- using an XML-based syntax
- supporting use of XML schema data types
- allowing anyone to make statements about any resource

Query and inference languages

The predominant query language for RDF graphs is SPARQL. SPARQL is an SQL-like language, and a recommendation of the W3C as of January 15, 2008.

An example of a SPARQL query to show country capitals in Africa, using a fictional ontology.

```
PREFIX abc: <nul://sparql/exampleOntology#> .
SELECT ?capital ?country
WHERE {
  ?x abc:cityname ?capital ;
     abc:isCapitalOf ?y.
  ?y abc:countryname ?country ;
     abc:isInContinent abc:Africa.
}
```

Other ways to query RDF graphs include:

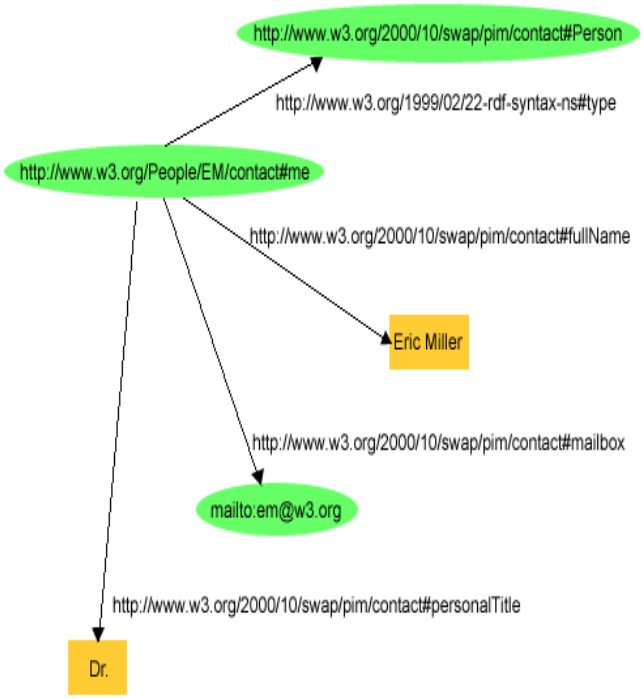
- RDQL, precursor to SPARQL, SQL-like
- Versa, compact syntax (non–SQL-like), solely implemented in 4Suite (Python)
- RQL, one of the first declarative languages for uniformly querying RDF schemas and resource descriptions, implemented in RDFSuite.
- SeRQL, part of Sesame
- XUL has a template element in which to declare rules for matching data in RDF. XUL uses RDF extensively for data binding.

##########

Examples

[edit]**Example 1: RDF Description of a person named Eric Miller**[17]

The following example is taken from the W3C website[17] describing a resource with statements "there is a Person identified byhttp://www.w3.org/People/EM/contact#me, whose name is Eric Miller, whose email address is em@w3.org, and whose title is Dr.".



An RDF Graph Describing Eric Miller

The resource "http://www.w3.org/People/EM/contact#me" is the subject.

The objects are:

- "Eric Miller" (with a predicate "whose name is"),
- em@w3.org (with a predicate "whose email address is"), and
- "Dr." (with a predicate "whose title is").

- The subject is a URI.

The predicates also have URIs. For example, the URI for each predicate:

- "whose name is" is http://www.w3.org/2000/10/swap/pim/contact#fullName,

- "whose email address is" is http://www.w3.org/2000/10/swap/pim/contact#mailbox,
- "whose title is" is http://www.w3.org/2000/10/swap/pim/contact#personalTitle.

In addition, the subject has a type (with URI http://www.w3.org/1999/02/22-rdf-syntax-ns#type), which is person (with URI http://www.w3.org/2000/10/swap/pim/contact#Person), and a mailbox (with URIhttp://www.w3.org/2000/10/swap/pim/contact#mailbox.)

Therefore, the following "subject, predicate, object" RDF triples can be expressed:

- http://www.w3.org/People/EM/contact#me, http://www.w3.org/2000/10/swap/pim/
- http://www.w3.org/People/EM/contact#me, http://www.w3.org/2000/10/swap/pim/contact#personalTitle, "Dr."
- http://www.w3.org/People/EM/contact#me, http://www.w3.org/1999/02/22-rdf-syntax-ns#type,http://www.w3.org/2000/10/swap/pim/contact#Person
- http://www.w3.org/People/EM/contact#me, http://www.w3.org/2000/10/swap/pim/contact#mailbox, em@w3.org

```
<urn:x-states:New%20York> <http://purl.org/dc/terms/alternative> "NY" .
```

In this example, "urn:x-states:New%20York" is the URI for a resource that denotes the U.S. state New York, "http://purl.org/dc/terms/alternative" is the URI for a predicate (whose human-readable definition can be found at here [18]), and "NY" is a literal string. Note that the URIs chosen here are not standard, and don't need to be, as long as their meaning is known to whatever is reading them.

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-
 rdf-syntax-ns#"
 xmlns:dcterms="http://purl.org/dc/terms/">
    <rdf:Description rdf:about="urn:x-states:New%20York">
            <dcterms:alternative>NY</dcterms:alternative>
```

Example 2: The postal abbreviation for New York

Certain concepts in RDF are taken from logic and linguistics, where subject-predicate and subject-predicate-object structures have meanings similar to, yet distinct from, the uses of those terms in RDF. This example demonstrates:

In the English language statement *'New York has the postal abbreviation NY'* , *'New York'* would be the subject, *'has the postal abbreviation'* the predicate and *'NY'* the object.

Encoded as an RDF triple, the subject and predicate would have to be resources named by URIs. The object could be a resource or literal element. For example, in the Notation 3 form of RDF, the statement might look like:

However, because of the restrictions on the syntax of QNames (such as dcterms:alternative above), there are some RDF graphs that are not representable with RDF/XML.

[edit]**Example 3: A Wikipedia article about Tony Benn**

To an English-speaking person, the same information could be represented simply as:
In a like manner, given that "http://en.wikipedia.org/wiki/Tony_Benn" identifies a particular
The title of this resource, which is published by Wikipedia, is 'Tony Benn'
resource (regardless of whether that URI could be traversed as a hyperlink, or whether the
However, RDF puts the information in a formal way that a machine can understand. The
resource is *actually* the Wikipedia article about Tony Benn), to say that the title of this
purpose of RDF is to provide an encoding and interpretation mechanism so
resource is "Tony Benn" and its publisher is "Wikipedia" would be two assertions that could
that resources can be described in a way that particular software can understand it; in other
be expressed as valid RDF statements. In the N-Triples form of RDF, these statements might
words, so that software can access and use information that it otherwise couldn't use.
look like the following:
Both versions of the statements above are wordy because one requirement for an RDF resource (as a subject or a predicate) is that it be unique. The subject resource must be

unique in an attempt to pinpoint the exact resource being described. The predicate needs to
<http://en.wikipedia.org/wiki/Tony_Benn> <http://purl.org/dc/elements/1.1/publisher>
be unique in order to reduce the chance that the idea of Title or Publisher will be ambiguous
"Wikipedia" .
to software working with the description. If the software
And these statements might be expressed in RDF/XML as:
recognizes *http://purl.org/dc/elements/1.1/title* (a specific definition for the concept of a title
<rdf:RDF
established by the Dublin CoreMetadata Initiative), it will also know that this title is different
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
from a land title or an honorary title or just the letters t-i-t-l-e put together.
xmlns:dc="http://purl.org/dc/elements/1.1/">

```
    <rdf:Description rdf:about="http://en.wikipedia.org/wiki/Tony_Benn">
        <dc:title>Tony Benn</dc:title>
        <dc:publisher>Wikipedia</dc:publisher>
    </rdf:Description>
</rdf:RDF>
```

The following example shows how such simple claims can be elaborated on, by combining multiple RDF vocabularies. Here, we note that the primary topic of the Wikipedia page is a "Person" whose name is "Tony Benn":

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:foaf="http://xmlns.com/foaf/0.1/"
 xmlns:dc="http://purl.org/dc/elements/1.1/">
    <rdf:Description rdf:about="http://en.wikipedia.org/wiki/Tony_Benn">
        <dc:title>Tony Benn</dc:title>
        <dc:publisher>Wikipedia</dc:publisher>
        <foaf:primaryTopic>
           <foaf:Person>
              <foaf:name>Tony Benn</foaf:name>
           </foaf:Person>
        </foaf:primaryTopic>
    </rdf:Description>
</rdf:RDF>
```

[edit]Applications

- Sigma - Application from DERI in National University of Ireland, Galway(NUIG).
- Creative Commons - Uses RDF to embed license information in web pages and mp3 files.
- DOAC (Description of a Career) - supplements FOAF to allow the sharing of résumé information.
- Enterprise Architect: MDG Technology for ODM (ODM supports RDF and OWL).
- FOAF (Friend of a Friend) - designed to describe people, their interests and interconnections.
- Haystack client - Semantic web browser from MIT CS & AI lab.[19]
- IDEAS Group - developing a formal 4D Ontology for Enterprise Architecture using RDF as the encoding.[20]
- Microsoft shipped a product, Connected Services Framework,[21] which provides RDF-based Profile Management capabilities.

- MusicBrainz - Publishes information about Music Albums.[22]
- NEPOMUK, an open-source software specification for a Social Semantic desktop uses RDF as a storage format for collected metadata. NEPOMUK is mostly known because of its integration into the KDE SC 4 desktop environment.

- RDF Site Summary - one of several "RSS" languages for publishing information about updates made to a web page; it is often used for disseminating news article summaries and sharing weblog content.
- ResumeRDF - developed to express information contained in a personal Resume or Curriculum Vitae (CV) on the Semantic Web. This includes information about work and academic experience, skills, etc.
- Simple Knowledge Organization System (SKOS) - a KR representation intended to support vocabulary/thesaurus applications
- SIOC (Semantically-Interlinked Online Communities) - designed to describe online communities and to create connections between Internet-based discussions from message boards, weblogs and mailing lists.[23]
- Smart-M3 - provides an infrastructure for using RDF and specifically uses the ontology agnostic nature of RDF to enable heterogeneous mashing-up of information[24]
- Many other RDF schemas are available by searching SchemaWeb.[25]

Some uses of RDF include research into social networking. This is important because it could help governments keep track of undesirables. It will also help people in business fields understand better their relationships with members of industries that could be of use for product placement.[26] It will also help scientists understand how people are connected to one another.

RDF is being used to have a better understanding of traffic patterns. This is because the information regarding traffic patterns is on different websites, and RDF is used to integrate information from different sources on the web. Before, the common methodology was using keyword searching, but this method is problematic because it does not consider synonyms. This is why ontologies are useful in this situation. But one of the issues that comes up when trying to efficiently study traffic is that to fully understand traffic, concepts related to people, streets, and roads must be well understood. Since these are human concepts, they require the addition of fuzzy logic. This is because values that are useful when describing roads, like slipperiness, are not precise concepts and cannot be measured. This would imply that the best solution would incorporate both fuzzy logic and ontology.[27]

RDF Schema

RDF Schema (variously abbreviated as **RDFS**, **RDF(S)**, **RDF-S**, or **RDF/S**) is a set of classes with certain properties using the RDF extensible representation language, providing basic elements for the description of ontologies, otherwise called RDF vocabularies, intended to

structure RDF resources. These resources can be saved in a triple store to reach them with the query language SPARQL.

Main RDFS constructs

RDFS constructs are the RDFS classes, associated properties and utility properties built on the limited vocabulary of RDF.

[edit]**Classes**

- **rdfs:Resource** is the class of everything. All things described by RDF are resources.
- **rdfs:Class** declares a resource as a class for other resources.

A typical example of an rdfs:Class is foaf:Person in the Friend of a Friend (FOAF) vocabulary. An instance of foaf:Person is a resource that is linked to the class foaf:Person using the rdf:type property, such as in the following formal expression of the natural language sentence : 'John is a Person'.

ex:John rdf:type foaf:Person

The definition of rdfs:Class is recursive: rdfs:Class is the rdfs:Class of any rdfs:Class.

The other classes described by the RDF and RDFS specifications are:

- **rdfs:Literal** – literal values such as strings and integers. Property values such as textual strings are examples of RDF literals. Literals may be plain or typed.
- **rdfs:Datatype** – the class of datatypes. rdfs:Datatype is both an instance of and a subclass of rdfs:Class. Each instance of rdfs:Datatype is a subclass of rdfs:Literal.
- **rdf:XMLLiteral** – the class of XML literal values. rdf:XMLLiteral is an instance of rdfs:Datatype (and thus a subclass of rdfs:Literal).
- **rdf:Property** – the class of properties.

[edit]**Properties**

Properties are instances of the class rdf:Property and describe a relation between subject resources and object resources. When used as such a property is a predicate (see also RDF: reification).

- **rdfs:domain** of an rdf:predicate declares the class of the *subject* in a triple whose second component is the *predicate*.
- **rdfs:range** of an rdf:predicate declares the class or datatype of the *object* in a triple whose second component is the predicate.

For example, the following declarations are used to express that the property ex:employer relates a subject, which is of typefoaf:Person, to an object, which is of type foaf:Organization:

ex:employer rdfs:domain foaf:Person

ex:employer rdfs:range foaf:Organization

Given the previous two declarations, the following triple requires that ex:John is necessarily a foaf:Person, and ex:CompanyX is necessarily a foaf:Organization:

ex:John ex:employer ex:CompanyX

- **rdf:type** is a property used to state that a resource is an instance of a class.
- **rdfs:subClassOf** allows to declare hierarchies of classes.

For example, the following declares that 'Every Person is an Agent':

foaf:Person rdfs:subClassOf foaf:Agent

Hierarchies of classes support inheritance of a property domain and range (see definitions in next section) from a class to its subclasses.

- **rdfs:subPropertyOf** is an instance of rdf:Property that is used to state that all resources related by one property are also related by another.
- **rdfs:label** is an instance of rdf:Property that may be used to provide a human-readable version of a resource's name.
- **rdfs:comment** is an instance of rdf:Property that may be used to provide a human-readable description of a resource.

[edit]**Utility Properties**

- **rdfs:seeAlso** is an instance of rdf:Property that is used to indicate a resource that might provide additional information about the subject resource.

- **rdfs:isDefinedBy** is an instance of rdf:Property that is used to indicate a resource defining the subject resource. This property may be used to indicate an RDF vocabulary in which a resource is described.

[edit]RDFS Entailment

An entailment regime defines by RDFs (,OWL, etc.) not only which entailment relation is used, but also which queries and graphs are well-formed for the regime. The RDFS entailment is a

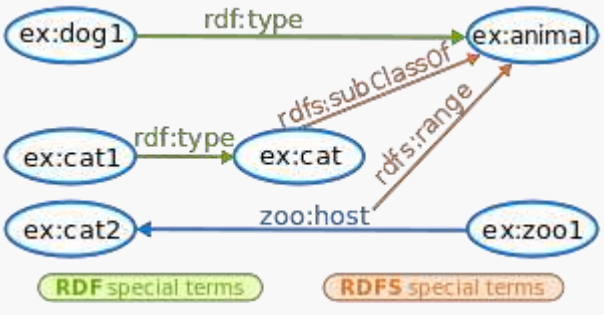
```
ex:cat      rdfs:subClassOf      ex:animal
```

Voila, the correct example:

For example, the following declares that 'Dog is an animal', 'Cat1 is a cat', 'Zoos host animals' and 'Zoo1 hosts the Cat2' :

```
ex:dog1      rdf:type          ex:animal
ex:cat1      rdf:type          ex:cat
zoo:host     rdfs:range        ex:animal
ex:zoo1      zoo:host          ex:cat2
```

But this graph is not well formed because the system can not guess that a cat is an animal. We have to add 'Cats are animals' to do a well-formed graph with :

In english	The graph
▪ Dog1 is an animal ▪ Cat1 is a cat ▪ Cats are animals ▪ Zoos host animals ▪ Zoo1 hosts the Cat2	

RDF/turtle

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex: <http://example.org/> .
@prefix zoo: <http://example.org/zoo/> .
ex:dog1    rdf:type        ex:animal .
ex:cat1    rdf:type        ex:cat .

ex:cat     rdfs:subClassOf ex:animal .
zoo:host   rdfs:range      ex:animal .
ex:zoo1    zoo:host        ex:cat2 .
```

If your triplestore (or RDF database) implements the regime entailment of RDF and RDFS, the SPARQL query as follows (the keyword "a" is equivalent to rdf:type in SPARQL):

```
PREFIX ex: <http://example.org/>
SELECT ?animal
```

```
WHERE
 { ?animal a ex:animal . }
```

Give the following result with *cat2* because the Cat's type inherits of Animal's type:

animal
<http://example.org/dog1>
<http://example.org/cat1>
<http://example.org/cat2>

RDFS Example

The following example demonstrates some of the RDFS facilities:

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xml:base="http://www.animals.fake/animals#">

<rdf:Description rdf:ID="animal">
        <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>

<rdf:Description rdf:ID="horse">
        <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
                <rdfs:subClassOf rdf:resource="#animal"/>
```

```
</rdf:Description>

</rdf:RDF>
```

Example Abbreviated

Since an RDFS class is an RDF resource we can abbreviate the example above by using rdfs:Class instead of rdf:Description, and drop the rdf:type information:

```
<?xml version="1.0"?>

<rdf:RDF

xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xml:base="http://www.animals.fake/animals#">

<rdfs:Class rdf:ID="animal" />

<rdfs:Class rdf:ID="horse">
    <rdfs:subClassOf rdf:resource="#animal"/>
</rdfs:Class>

</rdf:RDF>
```

XML Schema

An XML Schema describes the structure of an XML document.

XML Schema Example

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

XML Schema is an XML-based alternative to DTD.

An XML schema describes the structure of an XML document.

The XML Schema language is also referred to as XML Schema Definition (XSD).

What is an XML Schema?

The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD.

An XML Schema:

- defines elements that can appear in a document
- defines attributes that can appear in a document
- defines which elements are child elements
- defines the order of child elements
- defines the number of child elements

- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes

XML Schemas are the Successors of DTDs

We think that very soon XML Schemas will be used in most Web applications as a replacement for DTDs. Here are some reasons:

- XML Schemas are extensible to future additions
- XML Schemas are richer and more powerful than DTDs
- XML Schemas are written in XML
- XML Schemas support data types
- XML Schemas support namespaces

XML Schemas are much more powerful than DTDs.

XML Schemas Support Data Types

One of the greatest strength of XML Schemas is the support for data types.
With support for data types:

- It is easier to describe allowable document content
- It is easier to validate the correctness of data
- It is easier to work with data from a database
- It is easier to define data facets (restrictions on data)
- It is easier to define data patterns (data formats)
- It is easier to convert data between different data types

XML Schemas use XML Syntax

Another great strength about XML Schemas is that they are written in XML.

Some benefits of that XML Schemas are written in XML:

- You don't have to learn a new language
- You can use your XML editor to edit your Schema files
- You can use your XML parser to parse your Schema files
- You can manipulate your Schema with the XML DOM
- You can transform your Schema with XSLT

XML Schemas Secure Data Communication

When sending data from a sender to a receiver, it is essential that both parts have the same "expectations" about the content.

With XML Schemas, the sender can describe the data in a way that the receiver will understand.

A date like: "03-11-2004" will, in some countries, be interpreted as 3.November and in other countries as 11.March.

However, an XML element with a data type like this:

<date type="date">2004-03-11</date>

ensures a mutual understanding of the content, because the XML data type "date" requires the format "YYYY-MM-DD".

XML Schemas are Extensible

XML Schemas are extensible, because they are written in XML.
With an extensible Schema definition you can:

- Reuse your Schema in other Schemas
- Create your own data types derived from the standard types
- Reference multiple schemas in the same document

Well-Formed is not enough

A well-formed XML document is a document that conforms to the XML syntax rules, like:

- it must begin with the XML declaration
- it must have one unique root element
- start-tags must have matching end-tags
- elements are case sensitive
- all elements must be closed
- all elements must be properly nested
- all attribute values must be quoted
- entities must be used for special characters

Even if documents are well-formed they can still contain errors, and those errors can have serious consequences.

XML documents can have a reference to a DTD or to an XML Schema.

A Simple XML Document

Look at this simple XML document called "note.xml":

```
<?xml version="1.0"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

A DTD File

The following example is a DTD file called "note.dtd" that defines the elements of the XML document above ("note.xml"):

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

The first line defines the note element to have four child elements: "to, from, heading, body".
Line 2-5 defines the to, from, heading, body elements to be of type "#PCDATA".

An XML Schema

The following example is an XML Schema file called "note.xsd" that defines the elements of the XML document above ("note.xml"):

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">


<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>


</xs:schema>
```

The note element is a **complex type** because it contains other elements. The other elements (to, from, heading, body) are **simple types** because they do not contain other elements. You will learn more about simple and complex types in the following chapters.

A Reference to a DTD

This XML document has a reference to a DTD:

```
<?xml version="1.0"?>

<!DOCTYPE note SYSTEM
"http://www.w3schools.com/dtd/note.dtd">

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

A Reference to an XML Schema

This XML document has a reference to an XML Schema:

```
<?xml version="1.0"?>

<note
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

UNIT: IV

SYLLABUS

Web Ontology Language :OWL – Sub-Languages – Basic Notions -Classes- Defining and Using Properties – Domain and Range – Describing Properties - Data Types – Counting and Sets- Negative Property Assertions – Advanced Class Description – Equivalence – Owl Logic.

WEB ONTOLOGY LANGUAGE

rdfs:domain

For a property one can define (multiple) `rdfs:domain` axioms. Syntactically, `rdfs:domain` is a built-in property that links a property (some instance of the class `rdf:Property`) to a class description. An `rdfs:domain` axiom asserts that the subjects of such property statements must belong to the class extension of the indicated class description.

Multiple `rdfs:domain` axioms are allowed and should be interpreted as a conjunction: these restrict the domain of the property to those individuals that belong to the *intersection* of the class descriptions. If one would want to say that multiple classes can act as domain, one should use a class description of the `owl:unionOf` form. For example, if we want to say that the domain of the property `hasBankAccount` can be either a `Person` or a `Corporation`, we would need to say something like this:

```
<owl:ObjectProperty rdf:ID="hasBankAccount">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Person"/>
        <owl:Class rdf:about="#Corporation"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:ObjectProperty>
```

NOTE: In OWL Lite the value of `rdfs:domain` must be a class identifier.

4.1.3 rdfs:range

For a property one can define (multiple) `rdfs:range` axioms. Syntactically, `rdfs:range` is a built-in property that links a property (some instance of the class `rdf:Property`) to to either a class description or a data range. An `rdfs:range` axiom asserts that the values of this property must belong to the class extension of the class description or to data values in the specified data range.

Multiple range restrictions are interpreted as stating that the range of the property is the *intersection* of all ranges (i.e., the intersection of the class extension of the class descriptions c.q. the intersection of the data ranges). Similar to `rdfs:domain`, multiple alternative ranges can be specified by using a class description of the `owl:unionOf` form (see the previous subsection).

Note that, unlike any of the value constraints described in the section on class descriptions, `rdfs:range` restrictions are global. Value constraints such as `owl:allValuesFrom` are used in a class description and are only enforced on the property when applied to that class. In contrast, `rdfs:range` restrictions apply to the property irrespective of the class to which it is applied. Thus, `rdfs:range` should be used with care.

> NOTE: In OWL Lite the only type of class descriptions allowed as objects of `rdfs:range` are class names.

The OWL 2 Full semantics of Negative Property Assertions.

The Domain and Range of a negative property assertion

I suggest that the following entailment does **not** hold:

```
IF 'p' is a data property
AND 's p t' is NOT true
THEN t is a data value.
```

The problem is that it should be allowed to express that a triple 's p t' is not true, if t is known to be not a data value. The above entailment would be a semantic side effect of a negative property assertion, which might lead to heavy problems in certain occations.

More generally, nothing should be learnt about the origin of the LHS and the RHS of a negative property assertion, which isn't already known without.

Common Semantics

Axiomatic triples:

```
owl:NegativePropertyAssertion rdf:type rdfs:Class
owl:sourceIndividual rdf:type rdf:Property
owl:sourceIndividual rdfs:domain owl:NegativePropertyAssertion
owl:sourceIndividual rdfs:range rdfs:Resource
owl:assertionProperty rdf:type rdf:Property
owl:assertionProperty rdfs:domain owl:NegativePropertyAssertion
owl:assertionProperty rdfs:range owl:ObjectProperty
owl:targetIndividual rdf:type rdf:Property
owl:targetIndividual rdfs:domain owl:NegativePropertyAssertion
owl:targetIndividual rdfs:range rdfs:Resource
owl:targetValue rdf:type rdf:Property
owl:targetValue rdfs:domain owl:NegativePropertyAssertion
owl:targetValue rdfs:range rdfs:Resource
```

Note: The domain and range of 'owl:sourceIndividual' and 'owl:target*' have been deliberately chosen to be rdfs:Resource for the reason given in #The Domain and Range of a negative property assertion.

Negative Object Property Assertions

Syntax

```
x rdf:type owl:NegativePropertyAssertion
x owl:sourceIndividual s
x owl:assertionProperty p
x owl:targetIndividual t
```

Semantics

Main semantic condition:

```
IF
  (x,s) ∈ EXT_I(S_I(owl:sourceIndividual)),
  (x,p) ∈ EXT_I(S_I(owl:assertionProperty)),
  (x,t) ∈ EXT_I(S_I(owl:targetIndividual))
THEN
  x ∈ CEXT_I(S_I(owl:NegativePropertyAssertion)),
  p ∈ IOOP,
  (s,t) ∉ EXT_I(p)
```

Comprehension Principle: **[FIXME: necessary?]**

```
IF
  s, t ∈ IOT,
  p ∈ IOOP,
  (s,t) ∉ EXT_I(p)
THEN ∃ x:
  (x,s) ∈ EXT_I(S_I(owl:sourceIndividual)),
  (x,p) ∈ EXT_I(S_I(owl:assertionProperty)),
  (x,t) ∈ EXT_I(S_I(owl:targetIndividual))
```

Negative Data Property Assertions

Syntax

```
x rdf:type owl:NegativePropertyAssertion
x owl:sourceIndividual s
x owl:assertionProperty p
x owl:targetValue t
```

Semantics

Main semantic condition:

```
IF
```

```
    (x,s) ∈ EXT_I(S_I(owl:sourceIndividual)),
    (x,p) ∈ EXT_I(S_I(owl:assertionProperty)),
    (x,t) ∈ EXT_I(S_I(owl:targetValue))
  THEN
    x ∈ CEXT_I(S_I(owl:NegativePropertyAssertion)),
    p ∈ IODP,
    (s,t) ∉ EXT_I(p)
```

Comprehension Principle: **[FIXME: necessary?]**

```
  IF
    s ∈ IOT,
    t ∈ LV_I,
    p ∈ IODP,
    (s,t) ∉ EXT_I(p)
  THEN ∃ x:
    (x,s) ∈ EXT_I(S_I(owl:sourceIndividual)),
    (x,p) ∈ EXT_I(S_I(owl:assertionProperty)),
    (x,t) ∈ EXT_I(S_I(owl:targetValue))
```

What is Disjointness?

Two classes in an ontology are disjoint if they cannot share an instance, regardless of how the classes are interpreted.

For example:

1. There can be no animal (or anything else) that can be both an `Elephant` and a `Newt`.
2. If we said that the individual `tiny` is both a type of `Elephant` and a type of `Newt` and that `Elephant` and `Newt` are disjoint, then we would have an inconsistent ontology, i.e., one that can never be realised in a model.

3. Also, if we made a class `ElephantNewt` to be both a subclass of `Elephant` and a subclass of `Newt` (with the parents being disjoint), then we would have an unsatisfiable class `ElephantNewt`, i.e., in each of the ontology's models, there is never an instance of`ElephantNewt` (as it would have to be an instance of both `Elephant` and `Newt`, which isn't possible).
4. Similarly, if we say `Elephant SubClassOf: Animal` and then `Elephant DisjointWith: Animal`, then `Elephant` is unsatisfiable: all elephants are animals and no elephants are animals, a straight-forward contradiction, and thus there cannot be any `Elephant`.

How to Make Classes Disjoint

The most common way of expressing disjointness is with a disjointness axiom:

```
Elephant

DisjointWith: Newt
```

This just says directly that nothing can be both a type of `Elephant` and a type of `Newt`, and it applies to both named classes and unnamed ones.
Now disjointness statements aren't restricted to named classes: we can also state that class expressions have to be interpreted as disjoint sets, e.g., as follows:

```
(isPartOf some Nucleus)

DisjointWith: (isPartOf some Mitochondrion)
```

As a consequence of the above axiom, elements cannot be both part of the nucleus and part of the mitochondrion; a rather strong statement (though now it's explicit, it can be examined – the KR argument given above).

Additionally, there are several other ways to explicitly state disjointness, and it is also possible to imply disjointness – or both at the same time. We will now discuss some of these.

First, as an alternative to the above disjointness axiom on elephants and newts, we can say the following:

```
Elephant

SubClassOf: not Newt
```

This has exactly the same meaning and thus the same consequences as the first disjointness statement.

Next, we observe that, if we have two subclassses of our disjoint classes, then they are also disjoint: for example, `Indian Elephant` and `Great Crested Newt` will be disjoint when added to our animal ontology.

However, when we state that `African Elephant` and `Indian Elephant` are subclasses of`Elephant`, then they are not automatically disjoint; this has to be asserted or implied in some way.

As mentioned above, disjointness can be stated explicitly or implicitly, and we next discuss some simple ways of implicitly making classes disjoint. Take the classes

```
Class: Helium

SubClassOf: hasPart exactly 2 Proton

Class: Lithium

SubClassOf: hasPart exactly 3 Proton
```

As each and every `Helium` must have exactly two protons (no more and no less) and each and every `Lithium` must have exactly 3 protons, an atom cannot be both helium and lithium at the same time. Hence any explicit disjointness axiom on atoms is redundant if we model them in this way – and this safes us roughly 10K disjointness axioms in the 100 or so types of atom!

Similarly, if we have classes with the axioms

```
Class: A

hasPart at most 3  C

Class: B

SubClassOf: hasPart at least 4 C
```

then `A` and `B` are disjoint as there is no way an individual can fulfill both these criteria without contradiction. The next example is a non-obvious case of this situation:

```
Class: G

        SubClassOf: hasPart some  C

Class: H

        SubClassOf: hasPart only not C
```

Observe that `hasPart some C` means *has at least one part that is a C* and `hasPart only not C` means *has only parts that are not Cs*, which in turns means *has none/zero parts that are a C*. Hence the above two axioms also imply disjointness between G and H.

Finally, we want to mention that OWL 2 provides us with short-cuts in the forms of the`AllDisjoint` axiom and the `DisjointUnion` axiom. The latter makes a list of classes subclasses of the class on its left-hand-side, makes all classes in the list pairwise disjoint, and adds in a covering axiom for the left-hand-side being covered by the right-hand-side. For example,

```
Class: AminoAcid

        DisjointUnion Alanine, Cysteine, Aspartate, Glutamate,
        Phenylalanine, Glycine, Histidine, Isoleucine, Lysine,
        Leucine, Methionine, Asparagine, Proline, Glutamine, Arginine,
        Serine, Threonine, Valine, Tryptophan, Tyrosine
```

means that `AminoAcid` has all the subclasses listed, and that those subclasses are all disjoint. Also, and importantly, the listed amino acids are covering for `AminoAcid` (that is, any given amino acid must be an instance of one of the classes in the list).

Making Individuals Different

We can have a similar discussion about equality of individuals. Individuals may be the same unless we say or imply that they are different. We can do this, for example, with a`DifferentIndividuals` axiom and, as with disjointness, this will lead to expected but also possibly unexpected answers. Take the following example:

```
Individual: Harry

        Facts:

    hasFather Charles,

    hasFather James
```

If `hasFather` is stated to be functional, any individual may only hold one of this property to a distinct individual. However, without a different individuals axiom on `James` and `Charles`(or something implying it), they may or may not be identical, and thus the axioms above won't lead to an inconsistency. Adding a different individuals axiom on `James` and `Charles`(or something implying it) leads to an inconsistency.

UNIT: IV

SYLLABUS

Web Ontology Language :OWL – Sub-Languages – Basic Notions -Classes- Defining and Using Properties – Domain and Range – Describing Properties - Data Types – Counting and Sets- Negative Property Assertions – Advanced Class Description – Equivalence – Owl Logic.

WEB ONTOLOGY LANGUAGE

rdfs:domain

For a property one can define (multiple) `rdfs:domain` axioms. Syntactically, `rdfs:domain` is a built-in property that links a property (some instance of the class `rdf:Property`) to a class description. An `rdfs:domain` axiom asserts that the subjects of such property statements must belong to the class extension of the indicated class description.

Multiple `rdfs:domain` axioms are allowed and should be interpreted as a conjunction: these restrict the domain of the property to those individuals that belong to the *intersection* of the class descriptions. If one would want to say that multiple classes can act as domain, one should use a class description of the `owl:unionOf` form. For example, if we want to say that the domain of the property `hasBankAccount` can be either a `Person` or a `Corporation`, we would need to say something like this:

```
<owl:ObjectProperty rdf:ID="hasBankAccount">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Person"/>
        <owl:Class rdf:about="#Corporation"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:ObjectProperty>
```

NOTE: In OWL Lite the value of `rdfs:domain` must be a class identifier.

4.1.3 rdfs:range

For a property one can define (multiple) `rdfs:range` axioms. Syntactically, `rdfs:range` is a built-in property that links a property (some instance of the class `rdf:Property`) to to either a class description or a data range. An `rdfs:range` axiom asserts that the values of this property must belong to the class extension of the class description or to data values in the specified data range.

Multiple range restrictions are interpreted as stating that the range of the property is the *intersection* of all ranges (i.e., the intersection of the class extension of the class descriptions c.q. the intersection of the data ranges). Similar to `rdfs:domain`, multiple alternative ranges can be specified by using a class description of the `owl:unionOf` form (see the previous subsection).

Note that, unlike any of the value constraints described in the section on class descriptions, `rdfs:range` restrictions are global. Value constraints such as `owl:allValuesFrom` are used in a class description and are only enforced on the property when applied to that class. In contrast, `rdfs:range` restrictions apply to the property irrespective of the class to which it is applied. Thus, `rdfs:range` should be used with care.

> NOTE: In OWL Lite the only type of class descriptions allowed as objects of `rdfs:range` are class names.

The OWL 2 Full semantics of Negative Property Assertions.

The Domain and Range of a negative property assertion

I suggest that the following entailment does **not** hold:

```
IF 'p' is a data property
AND 's p t' is NOT true
THEN t is a data value.
```

The problem is that it should be allowed to express that a triple 's p t' is not true, if t is known to be not a data value. The above entailment would be a semantic side effect of a negative property assertion, which might lead to heavy problems in certain occations.

More generally, nothing should be learnt about the origin of the LHS and the RHS of a negative property assertion, which isn't already known without.

Common Semantics

Axiomatic triples:

```
owl:NegativePropertyAssertion rdf:type rdfs:Class
owl:sourceIndividual rdf:type rdf:Property
owl:sourceIndividual rdfs:domain owl:NegativePropertyAssertion
owl:sourceIndividual rdfs:range rdfs:Resource
owl:assertionProperty rdf:type rdf:Property
owl:assertionProperty rdfs:domain owl:NegativePropertyAssertion
owl:assertionProperty rdfs:range owl:ObjectProperty
owl:targetIndividual rdf:type rdf:Property
owl:targetIndividual rdfs:domain owl:NegativePropertyAssertion
owl:targetIndividual rdfs:range rdfs:Resource
owl:targetValue rdf:type rdf:Property
owl:targetValue rdfs:domain owl:NegativePropertyAssertion
owl:targetValue rdfs:range rdfs:Resource
```

Note: The domain and range of 'owl:sourceIndividual' and 'owl:target*' have been deliberately chosen to be rdfs:Resource for the reason given in #The Domain and Range of a negative property assertion.

Negative Object Property Assertions

Syntax

```
x rdf:type owl:NegativePropertyAssertion
x owl:sourceIndividual s
x owl:assertionProperty p
x owl:targetIndividual t
```

Semantics

Main semantic condition:

```
IF
  (x,s) ∈ EXT_I(S_I(owl:sourceIndividual)),
  (x,p) ∈ EXT_I(S_I(owl:assertionProperty)),
  (x,t) ∈ EXT_I(S_I(owl:targetIndividual))
THEN
  x ∈ CEXT_I(S_I(owl:NegativePropertyAssertion)),
  p ∈ IOOP,
  (s,t) ∉ EXT_I(p)
```

Comprehension Principle: **[FIXME: necessary?]**

```
IF
  s, t ∈ IOT,
  p ∈ IOOP,
  (s,t) ∉ EXT_I(p)
THEN ∃ x:
  (x,s) ∈ EXT_I(S_I(owl:sourceIndividual)),
  (x,p) ∈ EXT_I(S_I(owl:assertionProperty)),
  (x,t) ∈ EXT_I(S_I(owl:targetIndividual))
```

Negative Data Property Assertions

Syntax

```
x rdf:type owl:NegativePropertyAssertion
x owl:sourceIndividual s
x owl:assertionProperty p
x owl:targetValue t
```

Semantics

Main semantic condition:

```
IF
```

```
    (x,s) ∈ EXT_I(S_I(owl:sourceIndividual)),
    (x,p) ∈ EXT_I(S_I(owl:assertionProperty)),
    (x,t) ∈ EXT_I(S_I(owl:targetValue))
  THEN
    x ∈ CEXT_I(S_I(owl:NegativePropertyAssertion)),
    p ∈ IODP,
    (s,t) ∉ EXT_I(p)
```

Comprehension Principle: **[FIXME: necessary?]**

```
  IF
    s ∈ IOT,
    t ∈ LV_I,
    p ∈ IODP,
    (s,t) ∉ EXT_I(p)
  THEN ∃ x:
    (x,s) ∈ EXT_I(S_I(owl:sourceIndividual)),
    (x,p) ∈ EXT_I(S_I(owl:assertionProperty)),
    (x,t) ∈ EXT_I(S_I(owl:targetValue))
```

What is Disjointness?

Two classes in an ontology are disjoint if they cannot share an instance, regardless of how the classes are interpreted.

For example:

1. There can be no animal (or anything else) that can be both an `Elephant` and a `Newt`.
2. If we said that the individual `tiny` is both a type of `Elephant` and a type of `Newt` and that `Elephant` and `Newt` are disjoint, then we would have an inconsistent ontology, i.e., one that can never be realised in a model.

3. Also, if we made a class `ElephantNewt` to be both a subclass of `Elephant` and a subclass of `Newt` (with the parents being disjoint), then we would have an unsatisfiable class `ElephantNewt`, i.e., in each of the ontology's models, there is never an instance of`ElephantNewt` (as it would have to be an instance of both `Elephant` and `Newt`, which isn't possible).
4. Similarly, if we say `Elephant SubClassOf: Animal` and then `Elephant DisjointWith: Animal`, then `Elephant` is unsatisfiable: all elephants are animals and no elephants are animals, a straight-forward contradiction, and thus there cannot be any `Elephant`.

How to Make Classes Disjoint

The most common way of expressing disjointness is with a disjointness axiom:

```
Elephant

DisjointWith: Newt
```

This just says directly that nothing can be both a type of `Elephant` and a type of `Newt`, and it applies to both named classes and unnamed ones.
Now disjointness statements aren't restricted to named classes: we can also state that class expressions have to be interpreted as disjoint sets, e.g., as follows:

```
(isPartOf some Nucleus)

DisjointWith: (isPartOf some Mitochondrion)
```

As a consequence of the above axiom, elements cannot be both part of the nucleus and part of the mitochondrion; a rather strong statement (though now it's explicit, it can be examined – the KR argument given above).

Additionally, there are several other ways to explicitly state disjointness, and it is also possible to imply disjointness – or both at the same time. We will now discuss some of these.

First, as an alternative to the above disjointness axiom on elephants and newts, we can say the following:

```
Elephant

SubClassOf: not Newt
```

This has exactly the same meaning and thus the same consequences as the first disjointness statement.

Next, we observe that, if we have two subclassses of our disjoint classes, then they are also disjoint: for example, `Indian Elephant` and `Great Crested Newt` will be disjoint when added to our animal ontology.

However, when we state that `African Elephant` and `Indian Elephant` are subclasses of`Elephant`, then they are not automatically disjoint; this has to be asserted or implied in some way.

As mentioned above, disjointness can be stated explicitly or implicitly, and we next discuss some simple ways of implicitly making classes disjoint. Take the classes

```
Class: Helium

SubClassOf: hasPart exactly 2 Proton

Class: Lithium

SubClassOf: hasPart exactly 3 Proton
```

As each and every `Helium` must have exactly two protons (no more and no less) and each and every `Lithium` must have exactly 3 protons, an atom cannot be both helium and lithium at the same time. Hence any explicit disjointness axiom on atoms is redundant if we model them in this way – and this safes us roughly 10K disjointness axioms in the 100 or so types of atom!

Similarly, if we have classes with the axioms

```
Class: A

hasPart at most 3  C

Class: B

SubClassOf: hasPart at least 4 C
```

then `A` and `B` are disjoint as there is no way an individual can fulfill both these criteria without contradiction. The next example is a non-obvious case of this situation:

```
Class: G

        SubClassOf: hasPart some  C

Class: H

        SubClassOf: hasPart only not C
```

Observe that `hasPart some C` means *has at least one part that is a C* and `hasPart only not C` means *has only parts that are not Cs*, which in turns means *has none/zero parts that are a C*. Hence the above two axioms also imply disjointness between G and H.

Finally, we want to mention that OWL 2 provides us with short-cuts in the forms of the`AllDisjoint` axiom and the `DisjointUnion` axiom. The latter makes a list of classes subclasses of the class on its left-hand-side, makes all classes in the list pairwise disjoint, and adds in a covering axiom for the left-hand-side being covered by the right-hand-side. For example,

```
Class: AminoAcid

        DisjointUnion Alanine, Cysteine, Aspartate, Glutamate,
        Phenylalanine, Glycine, Histidine, Isoleucine, Lysine,
        Leucine, Methionine, Asparagine, Proline, Glutamine, Arginine,
        Serine, Threonine, Valine, Tryptophan, Tyrosine
```

means that `AminoAcid` has all the subclasses listed, and that those subclasses are all disjoint. Also, and importantly, the listed amino acids are covering for `AminoAcid` (that is, any given amino acid must be an instance of one of the classes in the list).

Making Individuals Different

We can have a similar discussion about equality of individuals. Individuals may be the same unless we say or imply that they are different. We can do this, for example, with a`DifferentIndividuals` axiom and, as with disjointness, this will lead to expected but also possibly unexpected answers. Take the following example:

```
Individual: Harry

        Facts:

    hasFather Charles,

    hasFather James
```

If `hasFather` is stated to be functional, any individual may only hold one of this property to a distinct individual. However, without a different individuals axiom on `James` and `Charles`(or something implying it), they may or may not be identical, and thus the axioms above won't lead to an inconsistency. Adding a different individuals axiom on `James` and `Charles`(or something implying it) leads to an inconsistency.

Syllabus

Semantic Web Tools and Applications :Development Tools for Semantic Web – Jena Framework - SPARI. – Querying Sematic Web – Semantic Wikis – Semantic Web Services – Modeling and aggregating social network data – Ontological representation of social relationships, Aggregating and reasoning with social network data. Understand semantic web basics, architecture and technologies.

Aggregating and Reasoning with Social Network Data

Representing Identity

- **URI (Universal Resource Identifier)**
- **Disambiguation (A and B are the same; There are two people called John Smith)**
- **OWL has the "sameAS" property**

0 **Equality**

0 **The property sameAs is reflexive, symmetric and transitive**

0 **Descriptive Logic vs. Rule based reasoners**

- **Rule based reasoners use forward chaining and backward chaining**
- **Descriptive logic is used for classification and checking for ontology consistency**

Ontological representation of social relationships

0 **FOAF is an example of an ontological representation of individuals**

0 **Eliminates the drawbacks of early social networks like Friendster, Orkut**

0 **The early social networks had centralized control and were difficult to manage**

0 **FOAF is distributed and has a rich ontology to characterize individuals**

Syllabus

Semantic Web Tools and Applications :Development Tools for Semantic Web – Jena Framework - SPARI. – Querying Sematic Web – Semantic Wikis – Semantic Web Services – Modeling and aggregating social network data – Ontological representation of social relationships, Aggregating and reasoning with social network data. Understand semantic web basics, architecture and technologies.

Aggregating and Reasoning with Social Network Data

Representing Identity

- **URI (Universal Resource Identifier)**
- **Disambiguation (A and B are the same; There are two people called John Smith)**
- **OWL has the "sameAS" property**

0 **Equality**

0 **The property sameAs is reflexive, symmetric and transitive**

0 **Descriptive Logic vs. Rule based reasoners**

- **Rule based reasoners use forward chaining and backward chaining**
- **Descriptive logic is used for classification and checking for ontology consistency**

Ontological representation of social relationships

0 **FOAF is an example of an ontological representation of individuals**

0 **Eliminates the drawbacks of early social networks like Friendster, Orkut**

0 **The early social networks had centralized control and were difficult to manage**

0 **FOAF is distributed and has a rich ontology to characterize individuals**